Problem solving from textbook examples

Thesis

Problem Solving from textbook examples

Sydney Ian Robertson, M.A., B.A.

Thesis submitted in partial fulfilment of requirements for Ph.D. in Psychology, February, 1994.

Human Cognition Research Laboratory
The Open University
Milton Keynes MK7 6AA
U.K.

Acknowledgements

I would like to thank the many people who gave their time and expertise to help me with this thesis. First, I would like to thank Stuart Watt in H.C.R.L. for re-writing the SOLO program from scratch. His patience, when I returned to him every now and then with a fresh nest of bugs for him to deal with, was nothing short of admirable.

I would also like to extend my gratitude to the staff and pupils of Blairgowrie High School in Perthshire and Denbigh High School in Milton Keynes. In particular I owe a debt of thanks to William Sutherland, head of the Mathematics department in Blairgowrie, and David Dalby, head of Mathematics in Denbigh. Many thanks also to Kathryn, Jon and Linda.

Finally, I owe a very great deal to Hank Kahney, my supervisor. Few students can have the good fortune to have someone who is prepared to lavish so many ideas, coffee and red ink on them.

Declaration

Parts of this thesis have been published as H.C.R.L. Technical Reports (96 and 97).

The Interpretation Theory presented in chapter 3 is a development of unpublished work by Hank Kahney of the Open University. He used the proportional analogy framework as the basis of a method of protocol analysis. The further development of the interpretation theory presented in chapter three is entirely my own work.

The interpretation theory is currently being used as the basis for a study into curriculum design, and to assess a Techniques Editor for novice Prolog programmers by the Department of Artificial Intelligence at Edinburgh University.

Abstract

There has been a great deal of research into students' use of examples when solving problems in textbooks. Much of this work has been within the framework of analogical problem solving (APS). Indeed many researchers believe they can build adequate models of how students learn and solve exercise problems by analogy to worked examples. In the first part of this thesis I argue that this view of problem solving from examples is inappropriate and often misleading. Most students learning a subject for the first time tend to *imitate* examples. Imitative Problem Solving (IPS) is a weak form of analogical problem solving. APS accounts assume that a solver has a representation of an earlier problem in memory. The difficulties involved are accessing that source problem and adapting it to solve the current one. IPS does not assume that the source is represented in memory, and even when the source example is available (as in textbook examples), the student may not understand it well enough to be able to adapt it to new situations.

The second part of the thesis presents an interpretation theory for analysing both texts and the behaviour of solvers using those texts to solve exercise problems.

The third part applies the interpretation theory to the solution explanation of a simple algebra word problem. Where an example problem fails to map directly onto an exercise problem, or where inferences have to be made to understand it, the solver will be unable to imitate the example and hence will have difficulties in proportion to the mapping inequalities between the two problems. That is, the interpretation theory allows us to predict precisely where solvers will have difficulty using an example to solve an exercise problem of the same type.

The final part presents experimental tests of these predictions. The results confirm that the interpretation theory analysis can correctly identify possible areas of difficulty for the student due to a) the way an example problem is structured, and b) the nature of the transfer task.

CONTENTS

Chapter 1 INTRODUCTION - A learning episode

1. Introduction

The research presented in this thesis is concerned with how people learn from texts. It deals specifically with how novices learn from examples, and how these examples are presented in teaching contexts. This chapter presents a learning scenario with two aims. The first is to clarify the goals of the thesis, and the second is to frame the issues to be considered. The scenario is a problem solving episode which occurred during the training of a subject learning a simple AI programming language. The learning episode will be referred to throughout the thesis; first, to identify the issues that will be dealt with; second, to act as a framework for discussing previous research; and third, as a yardstick against which to judge the merits of the research presented here. I shall attempt to show that the research reported here can make substantial contributions, at a finer grain of analysis than previously attempted, to our understanding of how novices tackle problems early in their attempts to learn a new domain of knowledge. Furthermore, the text analysis explained in this thesis shows how textbook writers can improve their presentation of material so as to increase the probability that students will understand the text at the same time as increasing the rate at which they learn from it. One of the main conclusions is that part of the 'blame' for any misunderstandings or misconceptions on the part of novices should be shifted from the novices themselves to their teachers.

Learning and problem solving from examples involve an interaction between the learner, the problem, and the domain represented by the text. Ideally, characterizing this

interaction requires information about the prior knowledge of the problem solver, a detailed analysis of the problem itself and of the text in which the problem is embedded. People do not readily learn what they do not understand. One of the main points that emerges from the studies reported here is that we should make few assumptions about what a learner is supposed to know. This approach partially shifts the burden for successful problem solving and learning from the learner to the teaching text.

2. A learning episode

In this section we shall examine the behaviour of a subject attempting to solve a problem in a programming language called SOLO. The subject's behaviour, the problem, and the nature of the text will be examined in some detail. First of all we shall look at some aspects of SOLO in general before looking at parts of the teaching material which the subject had read. The subject is then described before we go on to look at the nature of the problem she was asked to solve, and the verbal protocol she gave as she attempted to solve it.

2.1. The domain: The SOLO programming language

SOLO is an AI programming language designed by Marc Eisenstadt as part of a cognitive psychology course at the Open University (Eisenstadt, 1978/1983). Items in the SOLO database are in the form of *node-relation-node* triples and written in the form *node---relation--->node*. An example would be FIDO --- ISA ---> DOG. Triples can be added to a database using a procedure called NOTE, and removed from the database using a procedure called FORGET. For example, typing 'NOTE MARY --- KISSES ---> JOE' at the SOLO prompt would add that triple to the database and SOLO would respond:

OK...

```
MARY
  |
  |--- KISSES ---> JOE
```

If the user were then to type 'FORGET MARY --- KISSES ---> JOE' the triple would be removed from the database.

User-defined procedures are written by typing 'TO' followed by the procedure name and a parameter, if necessary. Parameters are enclosed in slashes thus: /X/. Each step in the procedure is numbered except for the final line where the user types 'DONE'. DONE lets SOLO know that the end of the procedure has been reached. An example of a user-defined procedure might be one where SOLO prints out that someone 'IS GREAT' when that someone is added as the parameter. In SOLO it would look like this:

SOLO: **TO PRAISE /X/**

...: 1 PRINT /X/ "IS GREAT"
...: DONE

Those parts in bold typeface indicate what the user has to type in. The parts in normal type are generated by SOLO itself.

The user can check to see if a triple is in the database by using the CHECK procedure. CHECK can be used in two ways: CHECK MARY --- KISSES ---> JOE would cause SOLO to respond 'PRESENT' if the triple MARY --- KISSES ---> JOE already exists in its database or 'ABSENT' if it does not. CHECK can also be used along with a wild-card represented by a question mark: CHECK MARY --- KISSES ---> ?. If a triple such as MARY --- KISSES ---> <some node> is present in the database, SOLO will respond by instantiating the value found at the second node of the triple, in this case JOE, to the wild-card variable and printing PRESENT: JOE.

When CHECK is used within a user-defined procedure, control-statements have to be added to tell SOLO what to do if the third node of a triple is either present in the database or absent. If at step 1 the user types in '1 CHECK /X/ --- LIKES ---> BEER,' SOLO generates two lettered substeps. Here they would be '1A IF PRESENT:' and '1B IF ABSENT:' The user has to specify what SOLO should do in either of these conditions. The control-statement EXIT causes SOLO to exit from the procedure at that point; CONTINUE tells SOLO to go on to the next numbered step. Here is an example of a procedure called PRAISE that CHECKs whether someone LIKES BEER and then either PRINTs something if the person does like beer or adds a new triple to the database if no match is found:

SOLO: **TO PRAISE /X/**
...: 1 CHECK /X/ --- LIKES ---> BEER
..... 1A IF PRESENT: PRINT /X/ "IS GREAT"; CONTINUE
..... 1B IF ABSENT: NOTE /X/ --- IS ---> TEETOTAL; EXIT
...: 2 PRINT "I REALLY ADMIRE" /X/ "'S TASTE"
...: DONE

If PRAISE BILL is then entered, and the triple BILL --- LIKES ---> BEER is present in the database, the program would first print BILL IS GREAT. The control statement CONTINUE passes control on to the next step, step 2, and SOLO would print "I REALLY ADMIRE BILL ' S TASTE". If the triple BILL --- LIKES ---> BEER did not exist in the database, then at step 1B the procedure would add BILL --- IS ---> TEETOTAL to the database and then EXIT without going on to step 2.

2.2. Recursion in the SOLO manual

This section examines the structure of the Appendix to the SOLO manual where recursion is introduced. The aim is to identify the goals of the writer based on the way recursion is presented and explained.

In the SOLO manual, a procedure known as INFECT is presented a number of times in various forms. What the writer is attempting to do is build on a simple procedure that adds a triple to the database (NOTE /X/ --- HAS ---> FLU) until eventually a recursive procedure is constructed that adds that triple to a *series* of nodes that are linked by the same relation. In other words, the INFECT procedure is increasingly elaborated until it includes a recursive call to itself in the final version.

The first INFECT procedure is very simple and has the effect of adding a triple to the database (for clarity each version of the INFECT procedure is numbered here, so this version is INFECT[1]).

Suppose we defined a procedure called INFECT as follows:

```
TO INFECT /X/
1 NOTE /X/ --- HAS ---> FLU
DONE
```

If SOLO's database contained the following description:

```
MARIA
   |
   |--- ISA ---> WOMAN
```

and we typed in:

```
SOLO: INFECT MARIA
```

SOLO would respond as follows:

OK...

MARIA

|

|--- ISA ---> WOMAN

|

|--- HAS ---> FLU

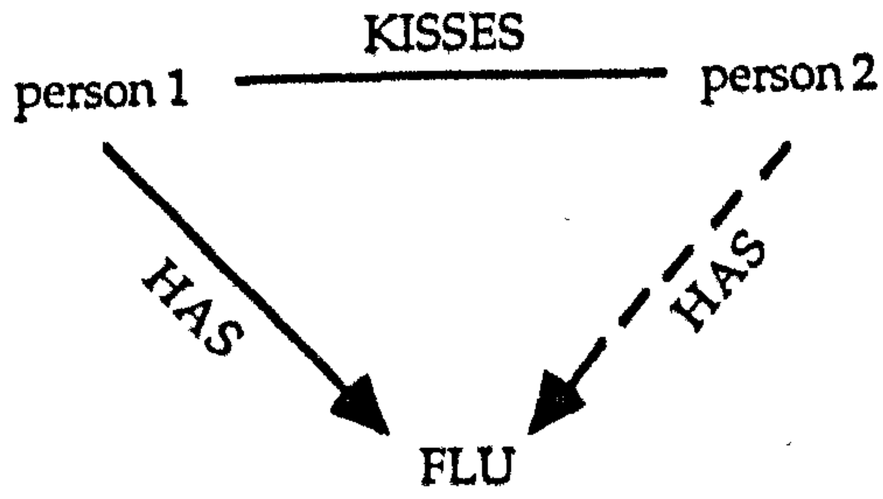
Notice that there is no statement of what the INFECT procedure is for. The writer assumes that the reader can infer what the program is designed to do.

A new version of the INFECT procedure (INFECT[2]) is then given along with a statement that 'if HARRY gets the FLU then JOAN gets it as well.' This is followed by the new definition:

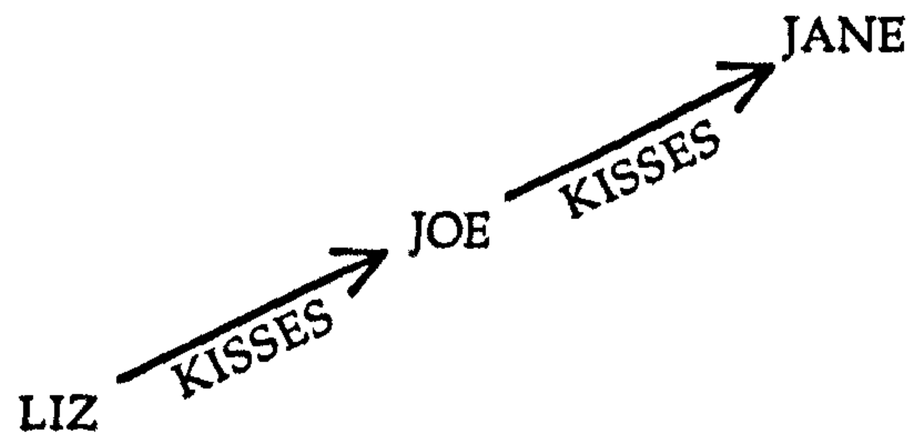
TO INFECT /X/
1 NOTE /X/ --- HAS ---> FLU
2 CHECK /X/ --- KISSES ---> ?
  2A IF PRESENT: NOTE * --- HAS ---> FLU; EXIT
  2B IF ABSENT: EXIT
DONE

(The asterisk after NOTE in the above definition is a variable which is instantiated to whatever matches the '?' in step 2).

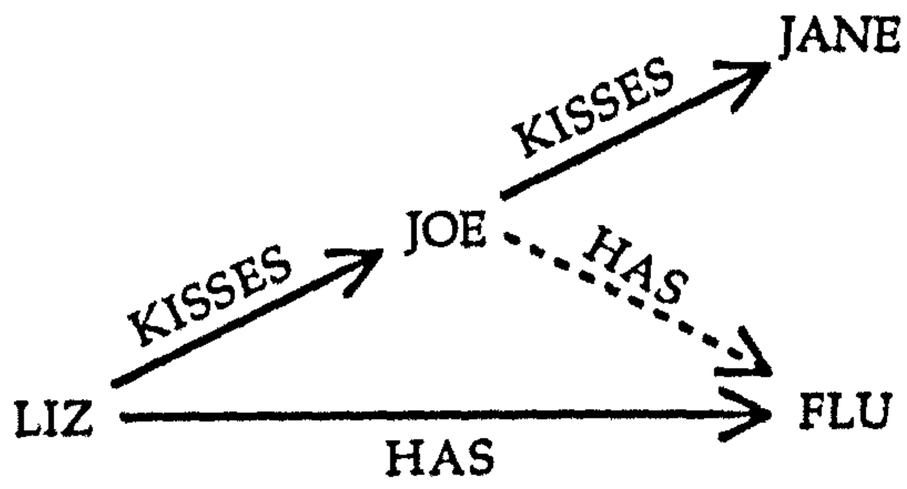
At this point the text invites the reader to say what would happen if INFECT HARRY is now typed in. An explanation of how SOLO would respond is then given which includes a diagram:



A hypothetical database is then presented diagrammatically:



The writer then asks what the above diagram would look like if INFECT LIZ is now typed in. The result is:



The explanation goes on to point out that JANE remains unaffected, whereas we would expect such a procedure to INFECT JANE as well.

The text goes on to say that 'we want [the procedure] to *keep on* happening,' and that the procedure would 'keep on happening' if one substep was altered. The text then gives the reader the chance to say what that alteration might be before providing it. In line 2A of the latest INFECT procedure (INFECT[3]) the NOTE * --- HAS ---> FLU has been replaced by INFECT *.

```
TO INFECT /X/
1 NOTE /X/ --- HAS---> FLU
2 CHECK /X/ --- KISSES ---> ?
  2A IF PRESENT: INFECT *; EXIT
  2B IF ABSENT: EXIT
DONE
```

INFECT[3] is then explained and three verbal analogies are introduced: a 'reserve pool' of football players waiting on a bench to go into action; the idea that procedures such as INFECT and NOTE are regarded as 'experts' at a particular task; and an analogy with passing a baton in a relay race. The text then presents a new database:

```
LIZ
    |
    |--- KISSES ---> JOE
JOE
    |
    |---KISSES ---> JANE
JANE
    |
    |---KISSES ---> HENRY
HENRY
    |
    |---SMOKES ---> CIGARS
```

The reader is then asked to imagine how SOLO would respond to INFECT LIZ given the new database. The answer is given and explained using the 'expert' analogy. The concept of recursion is further developed by pointing out two aspects: the concept of the recursive call (the procedure 'keeps on happening'); and the concept of the halting condition. In SOLO this is usually IF ABSENT: EXIT.

The first part of the SOLO Appendix is illustrated in figure 1.1[1]. In the figure each example problem statement is represented by A and the solution by B. Exercise and in-text questions are represented by C and the solutions by D. The rationale for this nomenclature is given in Chapter 3. The arrows show where the text 'takes' the reader. For example at row *b* the reader is asked how SOLO would respond if INFECT HARRY were typed in given the definition of INFECT[2] represented by B_2 in row *a*. The answer is then given (D_1) and the text goes on from that to explain how the most recent definition of INFECT would process INFECT HARRY. The 'e' superscript in B_2^e indicates that the solution is being explained. The explanation points the reader to some line diagrams. Diagrams, analogies, and such like are shown as B-prime as in B'_2 in row *b*.

In row *a* the A_1 represents the fact that there is no explicit statement of the problem to which the first INFECT program is a solution. The dotted line at row *c* shows where the

---

[1]*Figures 1.1, 1.2 and 1.3 are presented here to give a 'flavour' of how the interpretation theory presented in this thesis can be used to analyse texts. The interested reader may care to return to them after the full explanation of the theory in Chapter 3.*

reader would have to look to find the answer to question C2. In row *e* the explanation of SOLO's response (D_4^e) constantly refers to the three analogies given earlier (D'3 in row *d*).

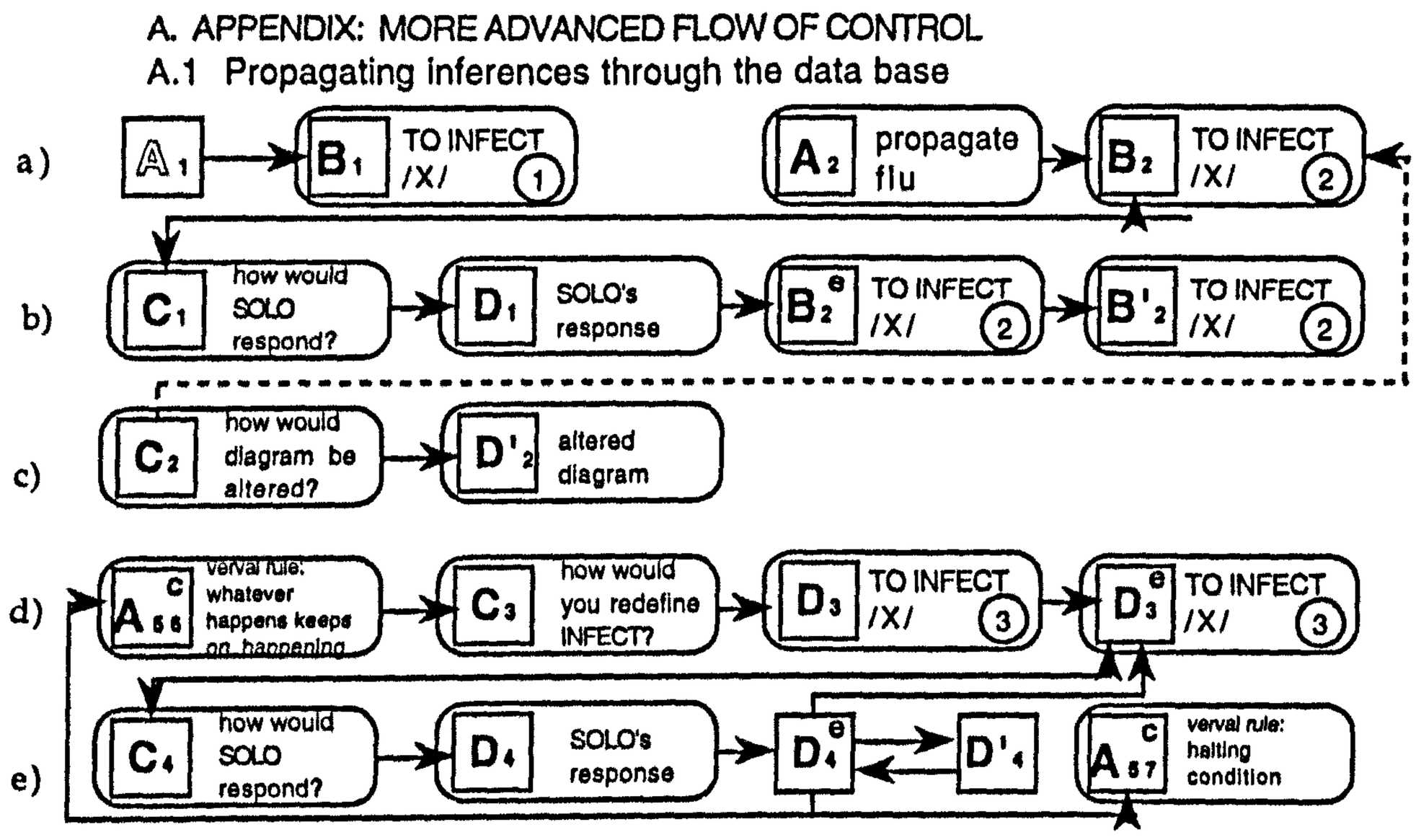


Figure 1.1. Representation of the text on recursion in the SOLO manual.

2.3. Iteration in the SOLO manual

The next few sections of the Appendix go on to discuss iteration and iteration along with recursion. They will be discussed only briefly since most of this part of the text is not directly relevant to the solution to the problem the subject was given.

Section A.2 of the SOLO Appendix goes on to discuss iteration which is explained by presenting an example solution called SUSS (with no statement of what the problem was). The limitations of the SUSS procedure are discussed leading on to a new version of SUSS incorporating a new SOLO construct FOR EACH CASE OF. The EACH CASE construct is used to find all matches for a particular node with more than one identical relation. For example, the following database contains several LIKES relations:

```
FIDO
  |
  |--- ISA ---> DOG
  |
  |--- HAS ---> FLEAS
  |
```

```
|--- LIKES ---> ROVER
|
|--- LIKES ---> BEER
|
|--- LIKES ---> SLEEPING
```

FOR EACH CASE OF FIDO --- LIKES ---> ? would identify the three nodes ROVER, BEER and SLEEPING that follow the LIKES relation. The iterative SUSS procedure given in the text is:

```
SOLO: TO SUSS /X/
...:1 PRINT "HERE ARE THE THINGS" /X/ "LIKES"
...:2 FOR EACH CASE OF /X/ --- LIKES ---> ?
.....2A: PRINT * "IS ONE OF THEM"
...:3 PRINT "I CAN'T THINK OF ANY MORE"
...: DONE
```

Given the current description of FIDO [above]
If we now typed in
SOLO: SUSS FIDO
SOLO would respond

```
HERE ARE THE THINGS FIDO LIKES
ROVER IS ONE OF THEM
BEER IS ONE OF THEM
SLEEPING IS ONE OF THEM
I CAN'T THINK OF ANY MORE
```

The text then presents two further examples and invites the reader to work out how SOLO would respond. The solutions are given along with an explanation.

The next section of the SOLO manual, section A.3, describes how FOR EACH can be nested within other FOR EACH constructs. This is illustrated with a new version of a procedure called CRAVETEST which was originally introduced in section 7.4 of the manual. Sections A.2 and A.3 are represented in figure 1.2.

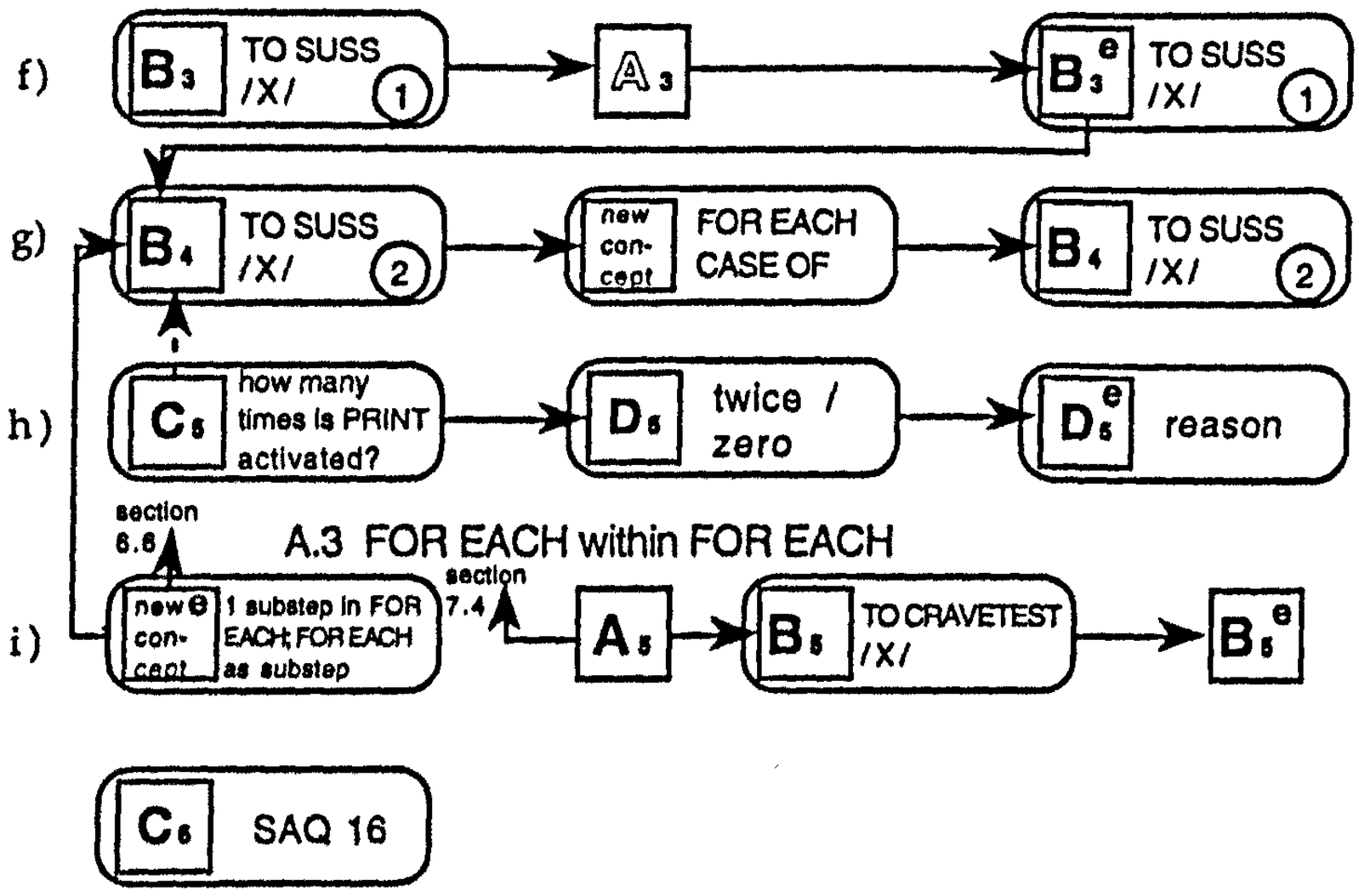


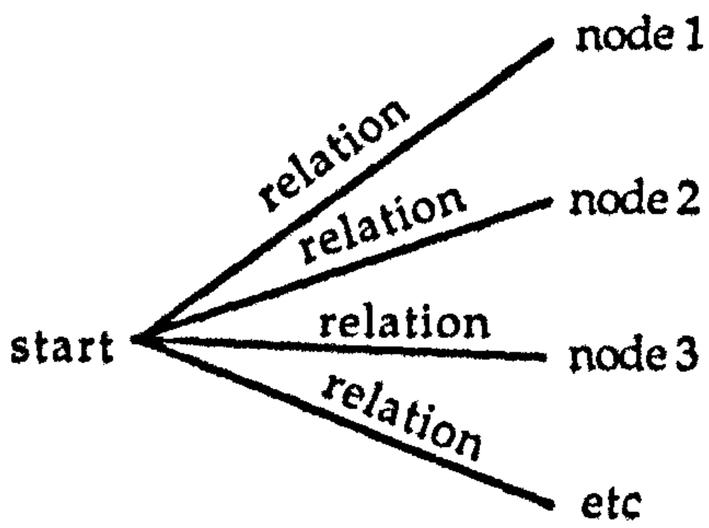
Figure 1.2. Representation of sections A.2 and A.3 of the SOLO manual.

Section A.4 explains that the CHECK node --- relation ---> ? construct is obsolete when the FOR EACH construct is first used. A new version of an earlier program called BOOZETEST is used to illustrate how the FOR EACH construct makes the CHECK construct redundant.

Section A.5 explains limitations on the use of variables. A new version of SUSS is used (SUSS[3]) to explain why. However, the SUSS program is an example of a *violation* of the rule restricting the use of variables.

Section A.6 explains how to edit a program within a substep. The EDIT procedure is explained by editing the latest version of a procedure called CRAVETEST which was introduced in an earlier part of the book. The final section compares iteration and recursion and shows how they can be used in conjunction. It begins with two diagrams representing iteration as 'fanning out' and recursion as 'propagating'. The diagrams used are:

iteration:



recursion:

start —relation→ node 1 —relation→ node 2 —relation→ node 3 —relation→ etc

The final definition of the INFECT procedure (INFECT[4]) is then given along with a brief explanation of how it works:

TO INFECT /X/
...: 1 NOTE /X/ --- HAS ---> FLU
...: 2 FOR EACH CASE OF /X/ --- KISSES ---> ?
..... 2A: INFECT*
...: DONE

The FOR EACH construct allows us to 'fan out' to discover everyone whom /X/ happens to kiss, while the recursive use of INFECT at sub-step 2A allows us to propagate the 'contagious' flu to everyone along a chain of KISSES relations.

The final part of the Appendix is represented in figure 1.3. The INFECT procedure in row *n* relates back to the original INFECT program at the beginning of the chapter (A2).

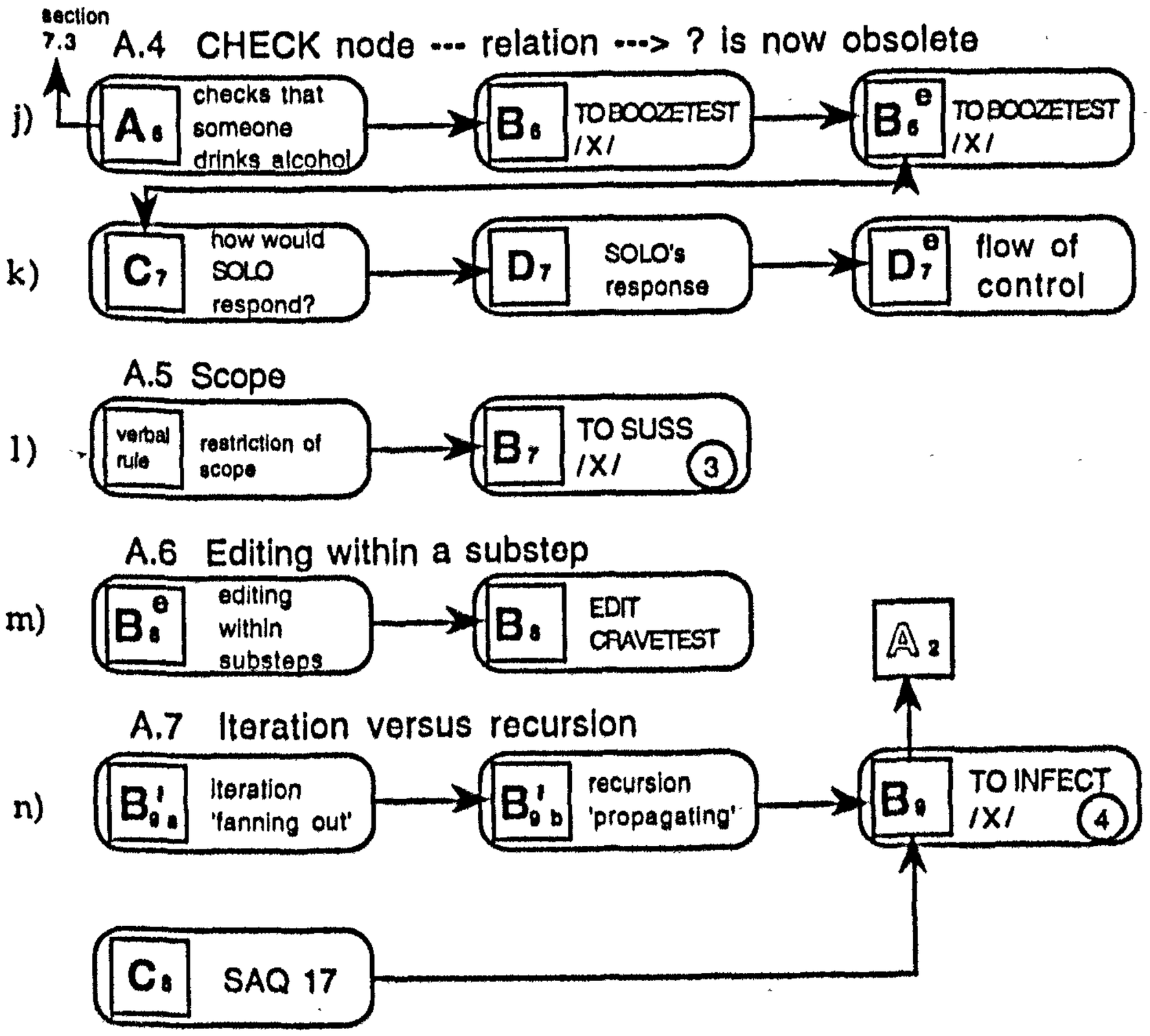


Figure 1.3. The final section of the SOLO Appendix.

2.4. The subject

Prior to solving the problem presented in the next section the subject had just finished the introductory SOLO manual. She did not know any other computer programming language. She had had a great deal of experience in learning from expository texts in the previous few years (e.g., tax law) involving complex problem solving tasks. It is therefore likely that she came to the SOLO training manual with certain expectations about how such texts should be structured. Indeed, there are several places in her study of the text where she said she would have preferred the text to give her the chance to 'think creatively'. She gives an example of this when she says:

> it would have been possible to put in some sort of sentence... question to encourage me to think about it. Like, say, something like, can you think of a problem that arises given what you've been told so far?

That is, she wanted to be allowed to reflect more on the text, or simulate example programs before having them explained. These comments are perhaps surprising given the large number of in-text questions in the manual asking the reader to predict how SOLO would respond to a particular piece of code.

During her learning of SOLO she showed further evidence of learning strategies in which she attempted to interpret the material she read (see Appendices A.3.1 and A.3.2). For example, on seeing the diagrams discussed above (pages 5 and 6) the subject made the following statement:

> I'm suddenly being thrown a diagonal line LIZ KISSES JOE KISSES JANE. I don't why they've been drawn diagonally, and I don't know if that has any significance particularly. But what it seems to be saying is, if we type in INFECT LIZ, then, because LIZ KISSES JOE, JOE will end up having flu, but it doesn't seem to take another step to have JANE having flu as well because JOE KISSES her.

The subject has therefore noted on studying the diagrams that JANE is not affected by the procedure. She has pre-empted the discussion in the text which goes on to point out that JANE should be INFECTed as well JOE. The subject notes:

> *reads paragraph 1 on page 80*
> Yes, in other words only one node can complete the asterisk variable thing, so we really need a loop of some sort so it keeps on coming back and adding on and not exiting

> until it's been through everybody that's been kissed by whatever person has been infected in the chain.

Again the subject pre-empts the discussion, and, in the process, generates the hypothesis that a 'loop' is needed. On reading the final recursive definition of INFECT[3] on page 80, she comments: 'In other words it loops.' She seems already to have generated her own model of recursion similar to the 'loop' model discussed by Kahney (1982).

When the text goes on to explain recursion with the three analogies of a 'reserve pool of football players', 'passing the baton in a relay race' and 'experts', the subject points out that she does not find them useful:

> I don't find the big paragraph right in the middle of page 80 any help at all. Mainly, I should think, because it doesn't seem strange that a procedure can activate itself. And the whole of that big paragraph starts off, 'If it seems somewhat strange that a procedure can activate itself'. It doesn't seem any stranger than the explanation that's given. Generally speaking I haven't found any of the analogies anywhere in this book any use at all, including the 'passing of a baton from one runner to the other in a relay race' for flow of control.

The first three excerpts above from her study of the training manual show that she attempted to draw inferences and generate hypotheses from the material as she read it. This kind of text processing is referred to as 'self-explanations' by Chi, Bassok, Lewis, Reimann & Glaser (1989) and Chi & Bassok (1989) or the 'reflective' understanding discussed by Hiebert & Lefevre (1986). These researchers would therefore classify her as a 'good student'.

2.5. The problem

When she had completed her study of the SOLO manual, the subject was presented with the following problem:

> Imagine a rambling four-storey house with a basement. A SOLO database describing part of the house looks like this:

```
ATTIC
        |
        | --- OVER ---> BEDROOM1
BEDROOM1
        |
```

```
        | --- ISA ---> ROOM
        |
        | --- OVER ---> LANDING
LANDING
        |
        | --- ISA ---> PLATFORM
        |
        | --- OVER ---> CUPBOARD
CUPBOARD
        |
        | --- OVER ---> LOUNGE
LOUNGE
        |
        | --- ISA ---> ROOM
        |
        | --- OVER ---> KITCHEN
KITCHEN
        |
        |--- ISA ---> ROOM
        |
        | --- OVER ---> BASEMENT
BASEMENT
        |
        | --- HAS ---> CONCRETEFLOOR
```

Now imagine that, after a severe frost, the water tank in the attic bursts.

Write a SOLO program called FLOOD to represent the fact that if the water tank in the attic bursts then the attic and anything below it get flooded.

Once the program has run, the items in the database should contain the following description:

```
ATTIC
        |
        | --- OVER ---> BEDROOM1
        |
        | --- IS ---> FLOODED
BEDROOM1
        |
        | --- ISA ---> ROOM
        |
        | --- OVER ---> LANDING
        |
        | --- IS ---> FLOODED
    LANDING
        |
        | --- ISA ---> PLATFORM
        |
        | --- OVER ---> CUPBOARD
```

```
        |
        | --- IS ---> FLOODED
etc.
```

The problem is isomorphic to the recursive INFECT problem on page 80 of the SOLO manual. An example database for the INFECT problem had the following form:

```
LIZ
   |
   |--- KISSES ---> JOE
JOE
   |
   |---KISSES ---> JANE
JANE
   |
   |---KISSES ---> HENRY
HENRY
   |
   |---SMOKES ---> CIGARS
```

The INFECT and FLOOD problems are isomorphic in that both involve a chain of nodes linked by the same relation name. In the INFECT problem, the chain relation looks like this:

LIZ $\xrightarrow{\text{KISSES}}$ JOE $\xrightarrow{\text{KISSES}}$ JANE $\xrightarrow{\text{KISSES}}$ HENRY

with each node linked by the KISSES relation. In the FLOOD problem, the chain relation looks like this:

ATTIC $\xrightarrow{\text{OVER}}$ BEDROOM1 $\xrightarrow{\text{OVER}}$ LANDING $\xrightarrow{\text{OVER}}$ CUPBOARD $\xrightarrow{\text{OVER}}$ LOUNGE $\xrightarrow{\text{OVER}}$ KITCHEN $\xrightarrow{\text{OVER}}$ BASEMENT

with each node linked by the OVER relation. The difference between the two lies in their 'surface structures'. That is, the relation and node names are different. Furthermore, the 'cover stories' for the two problems are not the same. The INFECT story involves the spread of 'flu through kissing and the FLOOD problem involves a burst water tank in an attic. On the surface, both problems are entirely different. Only in terms of their underlying structure are they similar.

The relevant solution to the INFECT problem was:

```
TO INFECT /X/
1 NOTE /X/ --- HAS ---> FLU
2 CHECK /X/ --- KISSES ---> ?
 2A IF PRESENT: INFECT *; EXIT
 2B IF ABSENT: EXIT
DONE
```

The solution to the FLOOD problem was:

```
TO FLOOD /X/
1 NOTE /X/ --- IS ---> FLOODED
2 CHECK /X/ --- OVER ---> ?
 2A IF PRESENT: FLOOD *; EXIT
 2B IF ABSENT: EXIT
DONE
```

The two solutions above have an identical structure. The value names in bold type in the INFECT program can be replaced with the corresponding values in the FLOOD program thus (⇒ means 'maps onto'):

INFECT ⇒ FLOOD

HAS ⇒ IS

FLU ⇒ FLOODED

KISSES ⇒ OVER

The FLOOD procedure will therefore work through the chain of OVER relations just as the INFECT program worked through the chain of KISSES relations. The side effect of the INFECT program is to NOTE that whatever node /X/ is instantiated to HAS FLU. The side effect of the FLOOD program is to NOTE that the particular part of the building represented by the variable /X/ IS FLOODED. When the program works through to the end of the chain and no more triples can be found by the CHECK at line 2, the program exits at line 2B.

2.6. Analysis of the protocol

The problem solving protocol that follows has been divided into a number of problem solving 'phases' that describe different aspects of the subject's behaviour. The phases are 1) 'problem understanding', 2) 'elaboration', 3) 'search', 4) 'comparison', 5) 'imitation', and 6) 'evaluation'.

2.6.1. *Problem understanding phases*

In the problem understanding phases, the subject is exploring the problem space trying to find a representation of the problem that will allow her to solve it. Understanding involves generating inferences from the text of the problem before attempting to write the code that will translate the problem representation into the form of the solution.

2.6.2. *Elaboration phases*

During this phase the subject explores some of the inferences she has generated in some detail. It includes an examination of how some of these inferences can be related to her knowledge of SOLO and the possible subgoals her program should contain.

2.6.3. *Search phases*

The search phase occurs when the subject reaches an impasse; that is, her problem solving is blocked and the subject searches through the SOLO textbook looking for an example or the explanation of a concept that might help in solving the problem.

2.6.4. *Comparison phases*

In the comparison phase, the subject has found a plausibly relevant example and compares it to the statement of the target problem. Here the subject is looking for similarities and differences between the example problem and the current one.

2.6.5. *Imitation phases*

An imitation phase occurs when the subject tries to use the example to help solve the current problem. To do this she copies the example problem by substituting different values from the target problem into the source.

2.6.6. *Evaluation phases*

In the evaluation phase, the subject is trying to assess the 'goodness' of her solution or partial solution to the problem. Evaluating a solution often means going back to the textbook to see if the solution fulfils some criterion the subject assumes to be necessary before the program will work

Some phases can be subsumed under others. *Elaboration,* for example, is an aspect of *problem understanding.* In understanding the problem the subject generates inferences and tries to see where these inferences lead in terms of the SOLO program. Similarly, when *evaluating* a solution the subject may *compare* it to other example solutions.

2.6.1. Problem understanding phase (1)

First the subject reads and then rereads the problem.

S1 (Subject): *Reads problem*

E (Experimenter): And if you look back over the problem question could you say which bits you're looking at.

S1: Yeah. Well, I'm going to read the whole lot over again.

Reads database out loud.

Next the subject classifies the problem.

S1: This is recursion rather than iteration. If that helps at all.

E: Why do you say that?

S1: Well, because it's not the fan-shaped thing, it's the line-shaped thing on page 91 of the book.

(*S1 gives this page number from memory - she does not refer to the book*)

E: What are you looking at?

S1: I'm looking at this again. I'm looking at the second page (*of problem*). Well, the first thing that I'm going to have to do is define a FLOOD procedure

The important aspect of this episode is *how* she manages to classify the problem. It appears that the subject already has some kind of representation of the structure of recursion problems in SOLO. This representation is based on a diagram of recursion as a chain relation. The subject refers to this without looking it up in the book. Furthermore, the subject infers from the structure of the problem's database that recursion should be used. The database can be construed as a succession of nodes (parts of a house) one *over* the other which become flooded as water passes through them. On page 91 of the textbook, the diagrammatic representations of databases that support iteration and recursion do not refer to a specific database. The diagrams are abstract. For example the 'line-shaped thing' the subject refers to is given on page 91 as:

start —relation→ node 1 —relation→ node2 —relation→ node 3 —relation→ etc

An example would be:

It seems that the subject therefore already understands the relation between such a representation and the form of the database. Classifying the problem is a first step to solving it, but not a guarantee that the subject necessarily understands recursion (although on her original reading of the text the subject showed evidence of having generated some form of 'loop' model of recursion). Although she may have an *abstract* idea of the concept of recursion, this does not guarantee that she can write a *specific* recursive procedure to solve the problem.

2.6.2. Elaborating the problem

In the next few episodes the subject switches her attention to working out what she can legitimately incorporate into the program based on the problem statement. There is information in the text of the problem that is not included as part of the database. For example, the problem statement mentions that there is a severe frost and that the water tank bursts. These problem features are not necessary to the program the subject is required to write. In the extract that follows, the subject feels that one of these aspects should be represented in the program she has to write, despite the fact that the problem requires her simply to write a program that generates ATTIC IS FLOODED, BEDROOM1 IS FLOODED, etc. as a side-effect. The problem statement mentions that the water tank in the attic bursts but there is no mention of a water tank in the database. The subject therefore infers that, since the problem states that the water tank bursts, that fact should form part of the FLOOD procedure.

> Em, and the thing that's sort of causing me immediate thoughts is: does my FLOOD procedure actually have to mention the fact that the water tank in the attic bursts, and I suppose it does.

She has therefore elaborated on the problem statement so that her problem model includes a representation of the water tank bursting.

In the next section the subject seems to be aware that her representation of the problem is not sufficiently constrained. She talks of the 'depth' the program should have, by which she means how much extra information it should have. The phrasing of the problem statement has led her to conclude that she should represent the fact that there is a water tank and that it bursts. Although the problem asks the solver to 'imagine that the water tank in the attic bursts' it also says: 'Write a SOLO program called FLOOD to represent

the fact that *if* the water tank in the attic bursts then the attic and anything below it gets flooded' (italics added). The word 'if' creates an inconsistency: the subject has to either write a program consistent with 'imagine that the water tank bursts' or one which is consistent with '*if* the water tank bursts'. The question was therefore badly phrased.

> I'm not quite sure exactly how much depth the program needs me to have. I mean does it need me to have somewhere in the database the fact that there is a tank in the attic, for instance?

In the next section the subject processes the problem again from the point of view of her 'elaborated' problem representation. That is, she looks at it under the assumption that it should include some representation of the water tank bursting. This representation leads to an impasse, however, since the subject wants to encode the fact that, *if* the water tank in the attic bursts, then anything below it gets flooded, but the fact that the water tank is in the attic is not in the database.

> Well, my problem is that, at the moment, the database doesn't contain the fact that there is a water tank in the attic, and therefore it's difficult to have... I mean, I was thinking, you see, of starting something like CHECK WATER TANK IN ATTIC BURSTS. Well, I mean, obviously it wouldn't say that, but, I mean, something that would check that sort of thing. But since the database at the moment doesn't have the fact that there is a water tank in the attic, then, em, that makes it a wee bit difficult, doesn't it?

2.6.3. Search phase

At this point the subject has reached a 'block' or impasse. In order to get out of it she looks back at the book. Her search, at first, is 'general', that is, she looks back through the textbook for help without looking for information on a specific concept. This is characterized as a 'search' phase. Her search quickly becomes more focused so that she ends up reading the book from the point of view of understanding the concept of recursion.

S1: OK, I'm going to look back at my little book to see if it can give me any ideas.

E: Which page are you looking at?

S1: I'm not looking at any particular page, I'm looking at page 90 but not for any particular reason because there's nothing on it that's of any particular use at the moment. Maybe it would help if I sort of worked backwards. Maybe it would help if I read the bit about recursion. That seems sensible, doesn't it?

2.6.4. Comparison phase

Here the subject finds an example of recursion (INFECT[3]) and reads it through with a view to understanding the concept of recursion. She contrasts this with the problem representation she had formed of the exercise problem solution and decides that her 'elaborated' problem representation is unnecessary. She explicitly contrasts what SOLO 'needs to know' with what a person would infer from the problem. That is, the SOLO model of the problem is different from her mental model of the situation described in the text of problem.

> OK. I don't have to have the fact... I'm looking on page 80, em... which has given a new definition of the INFECT procedure so that it keeps on recurring through it. So I probably... Right, OK. It was just to make... you see, em... I just thought it would be a bit clearer, not from the SOLO point of view, but simply clearer from the point of view of someone looking at it if the water tank was in it, but that doesn't really matter an awful lot, does it?
>
> So it's going to start off with, em, step1 NOTE ATTIC IS FLOODED without actually going a step behind that to explain why the attic is flooded in the first place. I think that's the only way I can start really, isn't it?

2.6.5. Imitation phase (1)

Having established the class of problems to which the test problem belongs, and having found an example of one on page 80 of the training manual, the subject imitates it by simply exchanging the values in the example solution with those in the target problem. The program she has written is successful.

> Em, and I'm still looking at page 80 because... why not? OK. Then it's going to be now... now I can see already that I'm gong to be putting in CHECK ATTIC is... OK. So it's not going to be NOTE ATTIC it's going to be NOTE /X/. I might as well get to there. Em, right, NOTE /X/ IS FLOODED
> CHECK /X/ OVER ?
> 2a IF PRESENT... IF PRESENT FLOOD *.
> *writes*
> Now this little thing I'm jotting down is simply copying what's on page 80 for the new INFECT procedure then I'm going to have to read it and discover that I've got to, sort of... well anyway...

writes

Right so far what I've written down is

TO FLOOD /X/
1 NOTE /X/ IS FLOODED
2 CHECK /X/ OVER ?
2a IF PRESENT: FLOOD *; EXIT
2B IF ABSENT: EXIT
DONE

The subject has mapped across the following:

TO INFECT /X/
↓ **map**
TO *FLOOD* /X/

1 NOTE /X/ --- HAS ---> FLU
↓ ↓ ↓ **map**
1 NOTE /X/ --- *IS* ---> *FLOODED*

2 CHECK /X/ --- KISSES ---> ?
↓ ↓ ↓ **map**
2 CHECK /X/ --- *OVER* ---> ?

IF PRESENT: INFECT *; EXIT
↓ **map**
IF PRESENT: *FLOOD* *; EXIT

IF ABSENT: EXIT
↓ **map**
IF ABSENT: EXIT

2.6.6. Evaluation phase

So far the subject has written the problem down on paper and has not yet entered the database or her program onto the computer. Had she done so and tried FLOOD ATTIC the program would have generated the correct response. As it is, the subject now enters an 'evaluation' phase in which she attempts to ascertain if the solution would work, particularly in terms of how the recursive call would work.

Now I just simply can't remember from the last time I did this whether that will take us right through everything, or whether that will simply take us through one case of it; so I'm just going to read my book a wee bit further on.

2.6.7. Search phase (2)

To check on the solution the subject once again looks through the training manual.

S1: This is page 82. And I'm looking on to the iteration, but that doesn't matter. I've decided that doesn't matter because we're not doing a fan thing here we're doing a line thing, but I may be wrong. In fact I may be seriously wrong.
reads p92.

2.6.8. Imitation phase (2)

In this episode, the subject maps part of the training problem solution (INFECT[4]) onto the exercise problem. Note that the subject makes a 'matching error' when she fails to change INFECT * to FLOOD *. This kind of error occurs when a problem is imitated but a value from the source problem (the training problem) is not adapted to the target problem (the exercise problem). Although this error is more in the nature of a slip of the tongue than an error, it does point up the extent to which the subject is imitating the source problem.

In fact, I'm now looking at page 92 and I suspect that what I want is something rather more like the INFECT thing on there which is NOTE /X/ IS FLOODED and then have a FOR EACH CASE OF /X/ OVER ? INFECT *. Right that's probably what I need, isn't it?

FOR EACH CASE OF /X/ --- KISSES ---> ?

↓ **map**

FOR EACH CASE OF /X/ --- *OVER* ---> ?

INFECT *

↓ **matching error**

*INFECT **

The subject in this episode has correctly inferred that a version of the exercise problem based on this example would also work.

2.6.9. Problem understanding phase (2)

In the next section the subject again rereads the problem and attempts to understand another aspect of it. In this case it is whether the attic gets flooded along with the rest of the building. In her first solution the attic did get flooded. Indeed, she changed NOTE ATTIC --- IS ---> FLOODED to NOTE /X/ --- IS ---> FLOODED earlier on, so this aspect of the problem did not cause any difficulty at that time. However, the wording of the question has led to confusion on the part of the subject, even though the problem states that when the program has run, ATTIC --- IS ---> FLOODED is one of the outcomes.

I'm sort of looking... I'm... I'm just basically, as you've gathered, copying the examples in the book, and the one on page 92, em... is...

E: What are you looking at now?

S1: I'm looking at the first page of the problem just to see exactly what the question's asking.

reads: "If the water tank in the attic bursts then everything below it gets flooded." The attic itself doesn't have to get flooded? Is that correct? Or does the attic too get flooded? Well, I mean, it... it implies that the rooms below it get flooded but the attic doesn't necessarily.

E: You can assume the attic gets flooded as well.

S1: Right. OK. I'm going to try my first problem and see what happens. This is because I really haven't a clue.

The subject is not confident that she has adequately understood the problem yet and decides to try the first solution. By this time, the subject has mapped two source problems with different structures onto the target problem. They are:

TO INFECT /X/
1 NOTE /X/ --- HAS ---> FLU
2 CHECK /X/ --- KISSES ---> ?
2A IF PRESENT: INFECT *; EXIT
2B IF ABSENT: EXIT
DONE

and

TO INFECT /X/
1 NOTE /X/ --- HAS ---> FLU
2 FOR EACH CASE OF /X/ --- KISSES ---> ?

2A: INFECT *
DONE

Both source problems would work as models. Whether or not the subject knows this is unclear.

2.6.10. Imitation phase (3)

The subject has now typed in the database and the first line of the solution. She once again looks at the example on page 80. This time she compares it to the solution she has already written and decides that it will work.

> Em, NOTE /X/ IS FLOODED. Back. Back to... what page was I on, page 52 or something? Page 80, I'm on page 80.
> *reads*
> Yeah, okay I think the first example is probably going to work, isn't it?

The subject types in the program she has written and then FLOOD ATTIC. The program works.

Figure 1.4 shows the path the subject made in the textbook. The parts of the Appendix which the subject ignores are blacked out. The large C box represents the test question (the FLOOD problem). The D box represents the target solution. In the first part of her search phase she finds INFECT[3] almost immediately (the dark line marked '1'). She then uses that solution to generate a solution to the FLOOD problem (line number 2). However, she is still unsure if the solution will work so she turns to page 82 which has an explanation of the INFECT[3] procedure (line number 3) but she spends very little time on that page and skips over the sections on iteration until she comes to the final INFECT[4] procedure. From there she goes back to the problem statement (line number 5) and types in the database and her original solution. Notice that the only parts she looks at in any detail are the two INFECT solutions, INFECT[3] and INFECT[4]. Apart from a very brief look at page 82, she does not read any of the textual explanations of the examples.

Although the text had spent five pages on an explanation of how to write a recursive procedure, the subject was still unable to generate a solution immediately even though the solution was isomorphic to the INFECT procedure. The subject therefore had a mental representation of recursion but could not remember an example in enough detail from her original reading of the text to write a recursive procedure that would solve the problem.

A second subject, whose protocol appears in Appendix A.3.6, did not recognize the FLOOD problem as an example of recursion. The subject looked through the manual for examples of how CHECK could be used and for examples using subprocedures. Eventually he came across the final INFECT procedure (involving iteration and recursion) as he searched through the book. The explanation there referred back to the earlier INFECT procedure. The subject therefore went back to that earlier procedure and used it as a model. This subject showed no evidence of having learned the concept of recursion nor the procedure for writing one from a study of the manual.

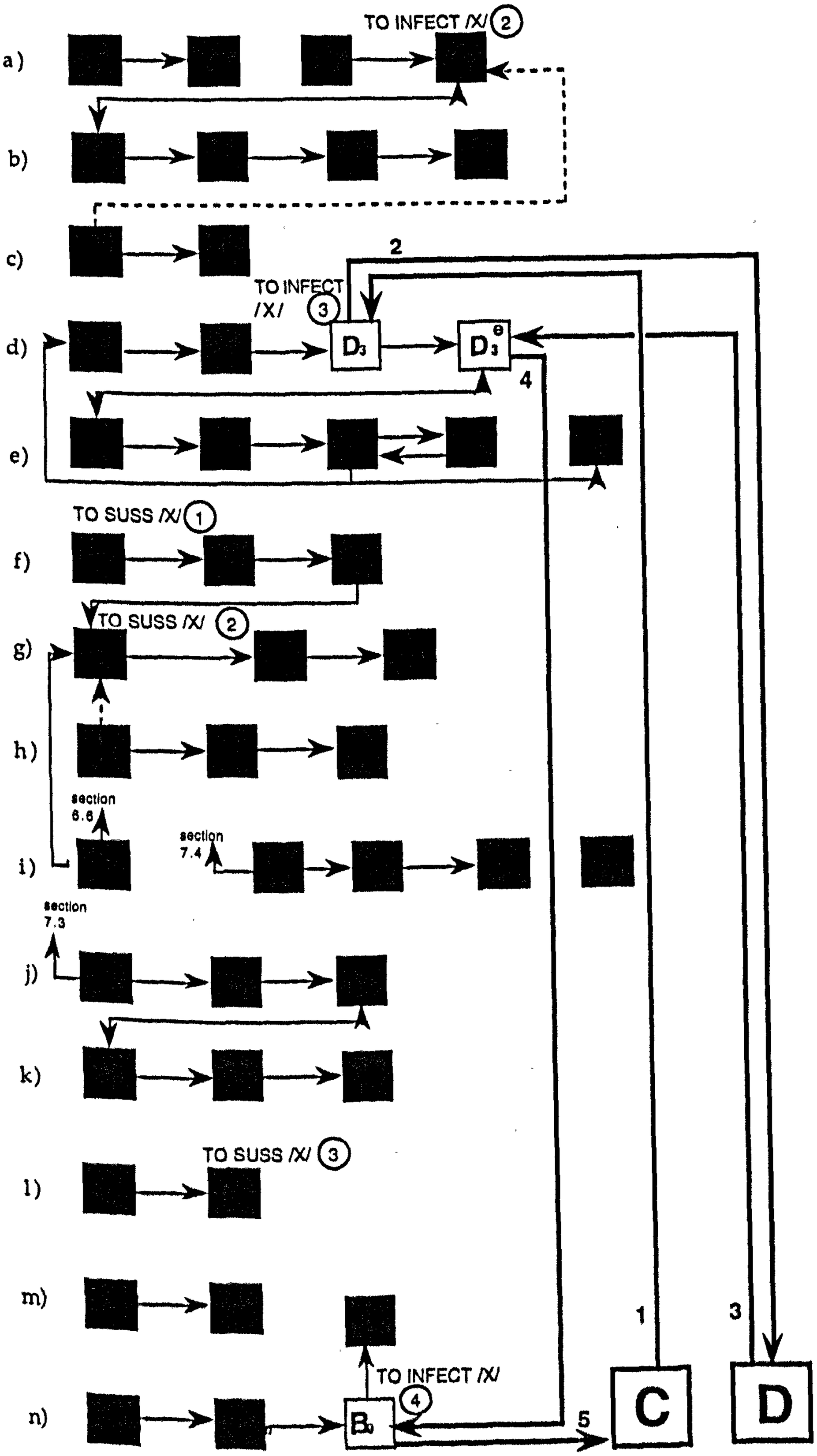


Figure 1.4. The subject's use of the textbook while solving the FLOOD problem.

3. The issues

Each of the phases of the problem solving scenario above throws up a number of issues. Each issue in turn has important implications for our understanding of the processes involved in problem solving and how the expository text can help or hinder those processes. The issues include: how the subject represented the problem; how she managed to classify it; what similarities she saw between the test problem and the INFECT[3] example; how she used the example to solve the problem; how she used the text itself; and why the text failed to equip her with the knowledge necessary to solve the problem without her having to search through the textbook. These issues are explored briefly here and in more depth in subsequent chapters.

3.1. Problem representation

In the first problem understanding phase, the subject simply read the statement of the problem and then reread parts of it. To solve problems from examples requires some kind of representation of the test problem, as well as of the examples and the concepts in the subject domain. The first issue therefore is how we can characterize the kind of representation of the problem statement the solver generates from reading it. The second concerns what assumptions we can make about how well solvers understand the domain from reading a text and the examples it provides.

One further related issue is the kind of elaborations solvers make of the problem. This depends on their prior knowledge. General world knowledge will allow them to generate a number of inferences based on the situation described in the text. Domain-specific knowledge allows them to generate inferences about how the situation described in the problem relates to concepts in the domain and any known procedure for solving this type of problem. The ability to generate domain-general and domain-specific inferences will determine how well a problem is understood.

When categorizing the problem the subject refers to two diagrams which she called the 'line-shaped thing' and the 'fan-shaped thing'. These are pictorial representations of the types of database on which iteration and recursion can work. Diagrams such as these form a 'bridge' between the presumed prior knowledge of the subject and some new concept which the text is trying to convey. For example, the concept of a chain of relations was illustrated by a line of different nodes linked by the same relation. In this way, the

diagram allows the reader to 'see'[2] what a chain of relations means. Such re-representations of a new concept form an 'intermediate representation' between the reader's prior knowledge and the new concept.

The verbal analogies used in the textbook to help explain novel concepts were not useful to this particular subject. For example, she says:

> Pages 40 and 41. I hate this automated telephone directory example. I don't think it makes anything clear at all. I can understand the automated telephone directory analogy only because I understand throughout the whole book what I'm required to do, and therefore I can understand the example. But it works that way round, and not the other way round which is presumably the intention.

However, expository analogies of this kind are often used in textbooks to allow the reader to extract the commonalities between the analogy and the novel concepts they are intended to illustrate. As such they also form a type of intermediate representation of new constructs. However, and most importantly, they can only work if the analogy is with concepts the reader already knows and understands. The telephone directory analogy seems to have failed in its usefulness with this particular subject. Her understanding of SOLO's relational database allowed her to understand the analogy of the automated telephone directory, which was the opposite effect from what the writer intended.

Another important issue is therefore the role such intermediate representations, both diagrams (including pictures, graphs, tables and the like) and verbal analogies, play in helping people understand expository texts.

3.2. Problem similarity

Having read the problem the subject successfully classifies it. What aspects of the problem (or her representation of it) allowed her to do so? Classifying the problem in terms of the underlying *principle* did not allow her to solve it immediately. She had to find some *procedure* for solving it. The procedure was instantiated in the examples she found.

---

[2]*The 'quasivisual nature of the phenomenology of comprehension' is discussed by Dennett (1991)*

There are a number of issues involved in understanding what makes two problems similar. An important distinction to make in this regard is between surface similarity and structural similarity. Two problems are superficially similar if they have the same 'cover story'. They are structurally similar if they involve the same underlying principle or solution procedure. The only example used in the SOLO textbook to explain recursion involved people infecting others with flu if they kissed them. The test problem given to the subject involved a water tank flooding all the floors beneath it. On the surface the two problems are different. The underlying procedure for solving them is identical, however.

A common goal of teaching texts is to explain how to solve problems that are superficially dissimilar but structurally similar. Although several pages were devoted to explaining how to write a specific program that INFECTs a series of people with flu, the text did not succeed in explaining how to solve other problems of the same type. That is, it failed to equip the reader with a procedure for solving other similar problems.

There are other aspects of problem similarity which merit discussion. For instance, in what ways do novices and experts see problem 'similarity'? What effects do variations between problems have on the people's ability to solve them? Is there some way of measuring these variations? When an example solution is used to help solve a test problem, how are the two 'mapped' - that is, what are the relations between the example and the exercise problem? The paradigm used in studying how analogues are seen as similar and how they are subsequently used has been that of analogical problem solving (APS). Much of Chapter 2 revolves around a discussion of whether APS is a useful paradigm to use in attempting to understand the processes involved in within-domain problem solving.

3.3. The role of hints

The subject in the above scenario found an example without being given any explicit hint to do so. One of the main difficulties in analogical problem solving is retrieving a relevant source example from memory that will help solve the current one. When solvers have formed some kind of representation of a target problem, they have to use that representation to access a plausibly relevant representation of some earlier problem from memory. When subjects are unable to solve the current problem, their representation of it is likely to be incomplete in some way. If it is incomplete, then it is because the underlying structure is not fully known or understood. For example, in the FLOOD problem the subject understands the problem statement. She does not understand immediately how to

translate her representation of the problem statement into SOLO code. That is, she does not know how the surface features of the problem relate to the underlying structure.

It is unlikely that a solver with an incomplete representation of either the target or the source will be able to access an earlier problem on the basis of structural similarities. For the novice, superficial similarities provide the main cues to retrieval of a plausibly relevant source analogue. When there are no surface similarities, a hint is often needed before subjects can retrieve a useful analogue. In the case of the subject described here, she found a relevant analogue by searching through the text for an example of recursion. She did not access the INFECT example through being given an explicit hint or because she had a representation of it in memory. A hint would have made her search unnecessary and speeded up the solution process.

3.4. The role of examples

When the subject found a plausibly relevant source example in the textbook, all she had to do was to replace the particular values in the source program with the values in the target problem. That is, TO INFECT was replaced by TO FLOOD, KISSES was replaced by OVER, INFECT * was replaced by FLOOD *. The subject was able to identify these correspondences. This is not always the case. Some subjects in an experiment by Kahney (1982) were unable to identify the correspondences between the INFECT[3] program and an isomorph (the SHOOTUP program in which a gun is fired from above at various objects piled up on a table).

Just as subjects may find difficulties in finding correspondences between problems which are dissimilar in their surface features, so they may find even more difficulty if the underlying procedure has to be adapted in some way. When the example is isomorphic the subject simply has to identify the correspondences between the two problems and replace them. However, if the structure varies, then finding correspondences is no longer sufficient. *Adapting* the example requires the subject to *understand* it at a deeper level than that of the surface features.

By substituting values across from a target to a source, solvers become aware that certain aspects of a problem or procedure can vary. Making variables out of specific values allows the solver to go beyond the specific context of the problems and abstract out the salient structural properties. When this happens, the solver has begun to induce the problem's underlying schema and can apply it to further problems of the same type. When the subject had completed the FLOOD problem, she was presented with Kahney's (1982)

SHOOTUP problem. She solved it without difficulty and without reference to the textbook. In other words she had *learned* how to solve simple recursion problems in SOLO.

3.5. The role of the text

At various times the subject refers to the difficulties she has in understanding the problem. What happens when novices are faced with difficulties they are unable to resolve immediately? How do they use textbooks to get round these 'impasses'? In the 'search phases' the subject searches through the book for help in overcoming her difficulties. This search might involve a search through the book for anything that looks like it might be relevant, or guided by the need to clarify a specific concept. Whatever type of search the subject engages in, it is important to know what aspects of the text determine how well this search will succeed. Both subjects in the SOLO experiment looked at example solutions. Very rarely did they look at the surrounding text.

In the above scenario the subject finds the INFECT[3] example and copies it. In generating her first solution she does not refer to the text around the example. Only when she finds she is unsure whether her program will work does she glance at the text for more information about recursion. Naturally there will be individual differences in the strategies people have in using textbooks, and in their acquaintance with different types of expository text. An important issue is therefore whether expository texts have or should have a canonical form, and how well a particular text conforms to it.

Textbook writers have to make a number of assumptions about what a reader has learned at different points throughout the book. Although the subject had recently learned about recursion and could recognize the problem type, she was still unable to remember in detail how it worked. The subject was asked to solve three test problems in SOLO. The first (given above) involved recursion. The second involved iteration but also required the subject to integrate information from much earlier in the textbook. There was no isomorph to the second test problem in the textbook. This problem proved extremely difficult for both subjects, even though the information required to solve it had been given at various places throughout the book.

The third problem the subject was given involved both iteration and recursion (see Appendix A.3.3.4). The subject solved it immediately. Somehow the subject learned something during her two previous problem solving experiences. The final issue is how teaching texts enhance learning and the role examples play in achieving that end.

4. The structure of the thesis

The points raised in the scenario presented here are explored in more depth in Chapter 2. The use novices make of examples has been the field of study of researchers in analogical problem solving (APS). However, solving problems by analogy means going beyond the surface features of problems and understanding in what way analogues are related in terms of their underlying structure. Successful use of an analogy means recognizing that two problems share the same relational structure and being able to apply that structure to the current problem. APS is contrasted with imitative problem solving (IPS) which does not require a solver to recognize the relational structure. Many of the difficulties that have been identified in the APS literature can be explained parsimoniously if we assume that solvers do not always know what the underlying structure of analogues is.

IPS has important ramifications for the design of teaching texts. In order to examine the use novices make of examples in texts, Chapter 3 presents an interpretation theory for text, task and protocol analysis. It allows the structure of texts to be analyzed at different grain sizes; from the sequence of examples, explanations, intermediate representations, and so on that texts provide, down to the fine structure of individual example problems. The use of the interpretation theory to give a coarse-grained view of a text is then presented. The analysis (of a LISP computer programming book) identifies, even at a coarse grain, where the reader may have to make inferences to understand the explanations provided, particularly the explanation of the relation between a problem statement and its solution.

By analyzing individual problems at a finer grain, we can see what mappings an example solution provides between a problem's givens and the form of the solution. An analysis at this level also indicates where there are gaps in the explanation of a solution procedure. When such gaps occur, solvers will have difficulty understanding the solution structure.

The mapping between two analogues can also be analyzed in considerable detail. The analysis can indicate where inferences have to be made and where the mapping has to be adapted to the novel problem. Where adaptations have to be made to an example, the analysis reveals just how 'distant' the target problem is from the source. The theory therefore provides a metric of transfer between problems. Chapter 4 demonstrates how simple word problems and the explanations they include of the solution procedure can be analyzed to reveal how effective they are likely to be.

From such a fine-grained comparison of individual problems we can make predictions about exactly where in a problem novices are likely to have difficulty. Especially when

the example has to be adapted in some way to solve a novel problem. In Chapter 5 these predictions are tested in a series of experiments using the same problems as those analyzed in Chapter 4. The experiments also show how explanations of word problems can be improved and how the subjects' assessment of their own understanding of the explanations can be made more accurate.

The final chapter recapitulates the main arguments of the thesis. They are that the structure of the text, and in particular the example problems it provides, determines to a large extent the behaviour of the student who uses the text to solve new related problems. Where solvers appear to be using the complex relational structure exemplified in a training problem to solve an exercise problem, it is often the case that they are in fact unaware of the structure and are simply adapting the surface features of the example. They are not therefore solving problems by analogy to the structure of the example but are *imitating* it.

Chapter 2 ANALOGICAL PROBLEM SOLVING AND IMITATION

1. Introduction

In the scenario introduced in Chapter 1 there was an interaction between the problem, the solver's prior domain-specific and domain-general knowledge, and the textbook. The solver went through a number of phases in her problem solving and I pointed out some of the aspects of her problem solving behaviour which needed further explanation. In formal domains, such as the SOLO programming language, new concepts are often introduced and explained by the use of examples and students are expected to use them to solve further problems of the same type. It has long been recognized that examples are 'potent teachers' (Ross, 1989a) and that skill acquisition depends on the way examples are presented in textbooks. Much of the research into this field has been aimed at understanding how we solve new problems by analogy to previous examples. In other words analogical problem solving (APS) has been the general framework into which this kind of research has fitted. The goal of this chapter is to examine to what extent this research effort helps us understand the processes involved in using examples to learn how to solve new problems. I will argue that novices mostly solve problems by *imitation*, which is a weak form of APS. Section 2 provides definitions for both APS and Imitative Problem Solving (IPS), and the distinction is thereafter used to assess the results of research in this field.

Having read the problem, the subject immediately classified it as belonging to a particular problem category (recursion). She must therefore have had some understanding

of the concept of recursion. That is, her representation of the problem must have been such that it matched an internal representation of recursive problems in SOLO already in her memory. She also spent some time trying to understand the question in such a way that it would lead to a useful solution plan. In other words she was trying to understand the situation described in the text in a way that would help her form a representation of the problem in SOLO. The nature of problem representations and conceptual understanding is discussed in section 3 of this chapter.

How one retrieves a representation of an earlier problem based on a representation of the current one depends on how the original problem was encoded and what cues are available in one's representation of the current problem. One of the main difficulties in APS is retrieving an appropriate analogue. The subject in Chapter 1 did not recall a relevant analogue at all. She retrieved a suitable model by searching through the textbook guided by her original categorization of the problem. She must therefore have seen some kind of similarity between the target and the retrieved base problem. Section 4 examines the complex nature of problem similarity.

In the imitation phase the subject adapted the source to fit the target problem. Mapping and adapting an analogue are further sources of difficulty in APS. The solver has to identify those elements in the source that correspond to elements in the target. The more dissimilar these elements are, the more difficult this mapping will be. Furthermore, if there are variations in the underlying solution procedure solvers will have difficulty adapting it. The question of what features of a problem are perceived as similar to an earlier problem reflects a concern with two aspects of examples: those features of a category of problems that can vary - the surface features; and those features that are invariant - the structural features. The interplay between surface and structure, and how solvers access and use examples are discussed in section 5.

The subject had previously learned the programming language SOLO from a textbook that introduced a number of unfamiliar concepts, which were presented in a variety of ways. There were textual explanations, diagrams, pictures and analogies, as well as concrete examples of how the new concepts were used in SOLO programs. There is virtually an infinite number of ways in which such expository texts can be organized. The text organization and the specific examples used will have a strong effect on how well a student will be capable of solving later exercise problems. For example, although the subject classified the problem correctly, she was unable to solve the problem immediately. One reason for this is that she did not classify the problem according to the solution procedure used to solve such problems in SOLO but rather she identified the problem according to an abstract representation of recursive problems given in the text as a

diagram. Diagrams are often used to make perceptually explicit the interrelations between features of a concept or problem. They are a way of re-representing concepts to allow the student to abstract out the common structure between the original concept and its re-representation. Such 'intermediate representations' are a half-way house between the novel concept and the learner's prior knowledge. Some of the difficulties facing textbook writers in the way they organize material and introduce new concepts are the subject of section 6.

The subject's schema or partial schema for recursion was created from her previous reading of the textbook. It was subsequently activated by her reading of the problem. A schema for a category of problems is a fairly abstract representation. Training examples in a textbook are very specific. Where did this schema come from? One might be inclined to wonder how novices can possibly learn anything from imitating an example. Imitation has, however, a strong pedigree. It is much used by children in their play. It is also a sensible thing to do. If you have a goal and you have a model that achieves a similar goal, it makes sense to use the model, whether you are fully aware of the logical justification of the form the model takes or not. We have an innate sense of causality. We know that events must have causes (whether we can find the right ones is irrelevant to this argument). Similarly, we know that the form of a problem structure (such as a SOLO program) must have something to do with its function (what the program is designed to do). With experience of a problem type we are able to abstract out the commonalities between problems and generalize to new ones. This inductive reasoning ability is discussed in section 7.

Several models of analogical problem solving are discussed throughout this chapter. Many of these have been built into AI models of learning and problem solving by analogy. One of these, the PUPS program from Anderson, Boyle, Corbett, & Lewis (1990), is discussed in section 8. I will argue that such programs are often very good models of human analogical problem solving competence but that they are poor models of human analogical problem solving performance. The reason for this is that they are built on theories of APS which assume too much on the part of the solver.

In conclusion, I argue that textbook writers and APS theorists alike make too many assumptions about how novices use examples in expository texts. They may not remember or understand the structure of complex problems, and often cannot solve them without a great deal of effort. Writers should not, therefore, expect transfer of learning from one example to a new problem unless they provide a great deal of information to help the novice to do so. The burden of responsibility for learning is thus shifted from the novice to the text the novice uses.

2. Analogical Problem Solving and Imitative Problem Solving

Very often the simplest way of solving a problem is to think of a similar one we have solved in the past and use that solution. In expository texts writers make much use of analogies to explain new concepts, and they are often quite effective in helping students understand them. The flow of water, for instance, is traditionally used to explain the flow of current in electricity. Indeed, 'flow' and 'current' applied to electricity derive from that analogy. Rutherford made the structure of the atom more comprehensible by drawing an analogy to the structure of the solar system. When analogies are used in a pedagogical context, their aim is often to allow the student to abstract out the shared underlying structure between the analogy and the new concept or procedure the student has to learn. Writers hope that it can thereafter be used to solve new problems involving that shared structure. Similarly, in a particular subject area, students may be expected to solve exercise problems based on examples that are analogous in that they involve the same solution procedure. When an example is presented as an analogy to help a student solve a target problem, it is in the belief that the student can abstract out the underlying structure of the example and apply it to the new problem.

There is a fundamental flaw with the latter view of the role of examples in problem solving. An expert might be able to see two superficially dissimilar problems as analogous, but a novice is much less likely to see them as such (Chi, Feltovich, & Glaser, 1981). Just as there is a distinction between linguistic competence and linguistic performance, so there is a distinction between the fact that two problems are analogous and what novices *do* when given an analogous solution to help them solve a problem. Novices may be aware that the structure of an analogue will help them solve a new problem with the same structure (Gentner & Landers, 1985); that is, people have a certain analogical problem solving competence. However, I will argue that novices often do not solve problems by analogy for the simple reason that they do not know in what way they are analogous. Instead, they attempt to *imitate* the example in the *belief* that the example is analogous to the current problem. In other words their analogical problem solving performance should be distinguished from their competence.

2.1. Analogical Problem Solving and Imitative Problem Solving compared

There are many definitions of analogy. Most of them emphasize the mapping of a systematic or hierarchical *structure* from an old (source or base) problem to a current

(target) problem. The base problem is stored in long-term memory (LTM) and the difficulty in employing APS is a function of:

a) the likelihood of finding the base problem when it is needed, given the nature of the new problem, and
b) the solver's ability in transforming old solutions to facilitate solving the new problem.

APS therefore involves the manipulation of a *mental representation* that a solver has of a previous problem (Anderson & Thompson, 1989; Carbonell, 1983; Gentner & Stevens, 1983; Gick & Holyoak, 1983; Holland, Holyoak, Nisbett, & Thagard, 1986; Holyoak & Thagard, 1989; Johnson-Laird, 1989; Kintsch, 1986). Imitative problem solving (IPS) is a weak form of APS in which the solver, faced with a new problem, looks back to an earlier problem example in a textbook, and tries to map the *surface features* of the source onto the target. No assumptions are made, in this case, about what the solver understands.

Before going on to look in more detail at the difficulties solvers have in retrieving and adapting a source problem, we have to be clear about what is involved in both APS and IPS, and why the argument that novices solve problems by analogy is at worst circular, or at best makes unwarranted assumptions about what the solver is expected to know.

Holyoak has listed several steps involved in APS[1]. Each step depends on the previous one. They are:

1. *forming a representation* of the source problem and the target problem;
2. *accessing* 'a plausibly relevant analogue in memory' (Holyoak and Thagard, 1989a). This depends on the form of representation the solver has of the problems. If solvers are to access a previously encountered problem from memory, then the representation they form of the target must match that of the source. This point begs the question of what constitutes problem similarity;
3. *mapping* across corresponding elements in the source and target. Correspondences are features which appear to play the same role in the source and target. Narrow sections of pipe restricting the flow of water have to be seen as corresponding to

---

[1]*Holyoak lists different steps at different times. The 5 presented here are derived from different sources (Catrambone & Holyoak, 1989; Holyoak, 1984; Holyoak, 1985; Holyoak & Koh, 1987; Holyoak & Thagard, 1989a; 1989b)*

resistors in an electric circuit before an analogy between the flow of water and the flow of electricity in complex systems can be understood;

4. *adapting* the mapping, constrained by the shared underlying structure of the problems, to generate a solution in the target. This step is also known as analogical inference or transfer. Knowing what happens to water when it passes through two narrow sections of pipe one after the other should allow students to infer what will happen when electricity has to pass through two resistors arranged in serial;

5. *learning*. This takes the form of schema abstraction from comparing examples (either implicitly or explicitly). With experience in solving a type of problem, students abstract out the underlying structure and can thereafter apply it to new versions of the problem type without having to refer to a previous solution.

Mapping and adapting are not regarded as being strictly sequential. Mapping 'can be conducted in a hierarchical manner, the process may be iterated at different levels of abstraction' (Holyoak, 1985). Analogical problem solving assumes that the solver has a representation of the source in LTM, and that the underlying structure of the problem is understood well enough so that the solver can map it across and generate inferences in the target. Imitation, on the other hand, does not make these assumptions. It is a problem solving process which involves:

1. *forming a representation* of the target problem. To the extent that novices are unlikely to have a 'complete problem model' (Holland et al. 1986) this representation will be impoverished in some way. Similarly, novices may not have a 'complete' representation of a candidate source problem;

2. *accessing* an earlier example, either by being given a hint to do so or through some other form of reminding, such as some perceived similarity between it and the target. This similarity is likely to be in terms of the problems' surface features, since imitation makes no assumptions about how well a source problem is represented. When solving an exercise problem in a textbook, the solver is likely to access an earlier problem by referring back to the examples given in the same chapter. Alternatively, as in the case of the subject in Chapter 1, the solver may access examples through some perceived similarity between it and an *aspect* of the test problem;

3. *mapping* values across that either,

   a) appear to fill the same roles in both problems, as in APS, or

   b) are perceptually similar, whether they fill the same roles or not. In a story about a squirrel, bluebird and zebra, children tended to map the zebra to a horse in an earlier story because they were similar sorts of things even though the animals played *different* roles in both (Gentner & Toupin, 1986);

4. *inferring operators* that will reduce the difference between the source and the target problems. The solver is trying to do the same things with the new values as was done in the source. They are therefore employing a form of means-ends analysis. There is no assumption that the solver understands the underlying relational structure of the source. If, in an example problem, a $1^1/_2$ hour difference is expressed as $^3/_2$, a solver may adapt a problem involving a difference of 2 hours by changing it to $^4/_2$, even though there is no need to do so. The solver is not acting with an understanding of the overall relational structure. Instead, the solution procedure is carried out in a step-by-step fashion.

5. *learning* by induction. The solver begins to infer a relation between the form of a category of problems (e.g. the syntactic structure of recursive programs in SOLO) and their function (what the programs are designed to do). This, as in many models of APS, comes about through exemplar comparison and use.

By assuming that IPS is the preferred method of problem solving by novices (and often experts), we can begin to understand better some of the difficulties that have been documented in the APS literature. At the same time IPS avoids many of the assumptions built into the APS account.

2.2. Difficulties in analogical problem solving

Research in APS has taken a number of forms. A common framework is to present an example problem and then a distractor task or a delay followed by a test problem of the same type as the example. Alternatively, a variety of examples is presented followed by a test problem to assess the effect of different problem presentations on problem solving. From this research we have come to identify a number of the difficulties involved in APS. These include difficulties in:

1) retrieving a relevant source example without a hint (Gick & Holyoak, 1980; 1983);

2) retrieving a relevant source example, unless the salient structural features of problems are pointed out (Bassok, 1990; Catrambone & Holyoak, 1989; Holyoak & Koh, 1987);

3) abstracting out the underlying schema, unless several examples are presented (Catrambone & Holyoak, 1989; Gick & Holyoak, 1983);

4) solving a test problem even when the similarity of the concepts in an analogous problem is pointed out (Novick & Holyoak, 1991);

5) understanding the relationship between training problems and other problems of the same type (Conway & Kahney, 1987; Reed, Ernst, & Banerji, 1974);

6) adapting a solution to apply it to a 'distant' variant of an example problem (Holyoak & Koh, 1987; Novick & Holyoak, 1991; Reed, Dempster, & Ettinger, 1985);

7) understanding the range of applicability of a new concept, unless a large number of example problems are provided (Cooper & Sweller, 1987; Sweller, 1988; Sweller & Cooper, 1985; Sweller, Mawer, & Ward, 1983);

8) using an intermediate representation, such as a diagram or expository analogy (Gentner & Gentner, 1983; Issing, Hannemann, & Haack, 1989; Reed & Ettinger, 1987).

9) generating inferences, since expository texts do not support inferences in the way that narrative texts do (Britton, Van Dusen, Glynn, & Hemphill, 1990; Britton, Van Dusen, Gulgoz, & Glynn, 1989).

Some of this research effort has been taken seriously by textbook writers who have attempted to make their explanations more accessible to their readers. At the Open University, for example, the SOLO manual was designed for students from a wide variety of backgrounds and tried to take account of many of these difficulties (see Eisenstadt, 1983). It included exercise problems with hints to look at earlier examples or explanations, a number of intermediate representations to explain novel constructs, such as recursion and iteration, and test problems which were 'close' variants of the examples provided. Despite this effort students still had difficulty using the manual to solve problems.

2.3. Top-down accounts of analogizing and the circularity problem

Many of the theories of APS involve a concept-driven account of problem solving. Structural views of analogy see APS as mapping a system of relations from a base to a target. This can only occur when the solver has enough knowledge of the relational structure of the base domain to extract it and apply it to the target. At worst, there is a danger that the arguments used in explaining APS may become circular. This is particularly the case when structural models are used to explain how a source is retrieved from memory.

The 'strong' view of analogical access is that a suitable analogue is accessed in LTM through a higher-order relational structure shared with the target (Schank, 1982). An example of higher-order relations might be a shared causal or explanatory structure. There is little evidence that people can retrieve analogues on this basis (Gentner & Landers, 1985; Gentner & Rattermann, 1991; Gick & Holyoak, 1980; 1983; Reed, Ackinclose, & Voss, 1990; Reed, Dempster, & Ettinger, 1985; Schumacher & Gentner, 1988).

Furthermore, the strong view of analogical access seems rather pointless. If a solver is given a test problem and realizes that its structure is similar to that of an earlier problem in LTM, then the solver must already be aware of the underlying explanatory structure. If

the solver is aware of the underlying structure there does not seem to be any need to access an analogue; the higher-order relation (the explanatory structure) can be applied directly to the givens in the target to generate the solution. Using an underlying structure to access an analogue is therefore unnecessary. In the problem solving episode with which we began, the subject does indeed access the earlier problem through an abstract principle (recursion). However, the way she goes about retrieving a source example does not fit in with the strong view of analogical access since the subject could not remember an earlier problem in detail, she found one by searching through a manual; the principle of recursion itself may not have been fully understood; and she did not appear to know how the principle and the form of the solution in SOLO were linked.

'Weaker' structural views of analogical problem solving are more common but have their own conceptual problems. If solvers do *not* immediately recognize the structure of a test problem, then they may have to be reminded in some way of a relevant source problem (the nature of these remindings is the subject of later sections). If we now suppose that the solver is given a hint to use a particular example, then the solver is presumed to be able to access the underlying structure of the earlier problem and apply it to the new one. This is the principle-cueing view, in which an analogy is a vehicle for a known solution schema.

Examples of this view can be found in the experiments of Gick and Holyoak (1980; 1983) who used two problems known as the Fortress Problem and the Radiation Problem (from Duncker, 1945). In the Fortress problem, a tyrant rules a country from a fortress in the centre. Roads radiate out from this fortress with villages along the way. A general arrives with an army large enough to overthrow the tyrant and take the fortress. However, the roads are mined in such a way that a large army will cause the mines to explode. Small groups, however, can travel along the roads without setting off the mines. That is how the tyrant maintains his supply routes. In order to protect his army and the villages along the roads, the General divides up his army into small groups and positions them at the head of the roads. On his signal the groups all converge on the fortress simultaneously and the fortress is captured.

In the Radiation problem a surgeon has to operate on a patient with a tumour. He has a ray machine which can destroy the tumour but will also destroy the healthy tissue surrounding it. His solution is to reduce the intensity of the rays and position several machines around the patient with the rays focused on the tumour. The surgeon, therefore, uses a 'divide and converge' solution, as did the General in the Fortress problem.

Using various experimental manipulations, Gick & Holyoak attempted to establish the conditions under which transfer would occur or fail to occur. The two problems provide

examples of similar *structural* features which can be abstracted out - the so-called 'convergence' solution. Understanding them necessitates a representation of the problems at a level of abstraction which can be applied to both. Very few of the subjects (about 20%) who had been shown the Fortress problem were able to solve the Radiation problem. When they were given a hint to use the earlier problem the number of correct solutions went up to around 90%. The hint was non-specific: the subjects were told that the earlier problem would help but not *how* it would help.

According to the principle-cueing view, the hint allows the solver to access the solution schema and apply it to the Radiation problem. Solvers do not therefore have to engage in analogical reasoning. They are presumed to know the solution structure already. The hint simply tells them which schema to use; all they have to do is apply it to the new problem. The function of the analogue is therefore to provide the schema, otherwise it is irrelevant. The principle-cueing view sees problem solving from examples as a 'top-down' process.

The main difficulty with extrapolating from the results of the experiments of Gick & Holyoak is that the base domain is usually well understood. People understand the relations between the givens in the Fortress problem and the motivations of the protagonists. The difficulty comes in adapting it to the new problem. In novel domains which present examples as models for further problem solving we can make no such assumption about how well the student understands the example. Indeed we cannot assume that the solvers engage in analogical problem solving at all, since there is no guarantee that they understand the structure of the example.

An IPS account of how solvers adapt the Fortress problem to solve the Radiation problem assumes that people will tend to look back at the earlier problem and map (what appear to be) corresponding problem elements. For example, solvers may realize that the general and the surgeon occupy the same role, that the army and the rays fulfil the same function, and that the fortress and tumour are the 'targets' of both problems. The next stage is to try to apply the same operators in the target as in the source. The general divided his army; so the solver has to work out how the surgeon can divide his rays. The general sent out groups to the heads of the roads radiating out from the fortress; so the solver has to work out how the rays can be similarly arranged; and so on. This is a bottom-up process in which the solver takes each step at a time and builds up a solution. It does not assume beforehand that the underlying solution schema is known or can be abstracted out and applied. Rather, if the solver succeeds in imitating the earlier solution, the schema is generated as a by-product of the solution process.

2.4. Summary

Solving a problem by analogy assumes that the solver understands the underlying structure of the analogue. If the structure is understood well enough, then it can be adapted to solve another problem of the same type. Many of the studies of APS have shown that this form of transfer is rare. Subjects can often solve problems which are isomorphic to a source only if they are given a hint to use the source, and if they can identify the correspondences between the source and the target. In homomorphic problems (or 'distant' variants of an example problem) the solution often has to be adapted even when the surface features are the same. The evidence suggests that the likelihood that a solver can adapt a source to any great extent is remote.

These difficulties point to the fact that example problems in a new domain are often not well understood. If solvers succeed in solving versions of a problem that are very similar (*close variants* of the problem) and fail to solve ones which have to be modified (*distant variants*), however slightly, then they must be using a process that allows them to adapt a source without understanding it. This process is imitation. It allows solvers to solve problems which are 'literally similar' (Gentner, 1989), but will lead to errors or failure when the source has to be adapted to a more distant variant.

The difficulties in APS therefore hinge on how well base and target problems are represented in the minds of solvers - and how 'similar' those representations are to one another.

3. The problem of representation

In the problem solving episode in chapter 1, the subject began by trying to understand what the problem 'meant'. There were several aspects to this. First of all she read the problem twice. She then spent some time trying to find the constraints on the problem based on her understanding of the text: she did not know if there was any need to represent the fact that there was a water tank in the attic, nor whether she should start with something that CHECKs if the water tank bursts. A third aspect of her problem representation involved her knowledge of SOLO. She knew many of SOLO's concepts and had had practice at writing SOLO procedures and function definitions. She was also aware that her initial understanding of the problem was insufficient to solve it. At this point she looked back for an example *because* she did not have an adequate representation of the task she was required to do. On the other hand, she did know that the solution involved recursion. How can we characterize these types of representation?

3.1. Encoding the text of a problem

The subject generated a number of inferences from the problem statement. These included inferences about relevant bits of SOLO procedures for converting information in the problem into a SOLO program. For example, she generated the inference that 'the water tank bursts' should form part of the SOLO program and that this would be structured something like 'CHECK WATERTANK INATTIC BURSTS'.

Various representations can be derived from a textual presentation of a problem. First of all there are the individual words that compose the text. Understanding these comes through our semantic knowledge of the items in our mental lexicon. From the individual words and the context of the sentence, our overall understanding of the text of a problem is constructed. For problem solving to be successful, the solver has to generate all the necessary inferences in order to build a useful problem model. This, in turn, means that novices have to have enough domain-relevant knowledge to do so. Knowing what the text of a question means does not therefore entail an understanding of the problem. In this regard Kintsch (1986) has noted: 'all too often we seem to "understand" the text all right but remain at a loss about what to do [to solve a new problem].'

Kintsch (e.g. 1986; Nathan, Kintsch, & Young, 1992; Van Dijk & Kintsch, 1983) has argued that word problems require the solver to generate a number of different representations. Solvers first form a propositional representation of the text of such problems called a

textbase. From that they develop a representation of the situation described in the text. This is a mental model which he terms a *situation model*. The situation model is an elaborated one which includes inferences about the situation described in the problem statement. An example taken from colinear distance problems involves a vehicle that leaves from a particular point some time after another vehicle and eventually overtakes it. According to Nathan, Kintsch & Young (1992), 'the situation model reveals that they will have [travelled] equal distances at that point' (p. 333). They also add that,

> 'because of the added demands of inference making, readers will make inferences only when they seem necessary. Poor problem solvers will tend to omit them from their representations, and so they will *omit* the associated equations (supporting relations) from their solutions to story problems. Problem solvers who reason situationally will tend to include these inference-based equations' (p. 335).

A further representational form proposed by Kintsch and his co-workers is the *problem model* which includes formal knowledge, for example, about the arithmetic structure derived from the text, or the operating procedure constructed from information in the text[2]. The ability to make inferences from texts in order to derive a useful problem model depends on the relevant prior knowledge of the learner.

> The subject's protocol reveals that she had not yet settled on a fixed model of the situation represented in the text. She eventually used the *example* to constrain her understanding of the situation - that is, that the program doesn't have to represent the fact that there is a water tank in the attic. The example allowed her to reconstrue the current problem before going on to translate this new representation into SOLO code.

Kintsch (1986) argues that the text determines what problem model is constructed and how it is constructed. In particular the problem model is important for learning and the textbase model is important for remembering text. In a study of problem solving and retrieval of earlier problems, he found that recall of word problems that had already been solved was determined both by the properties of the textbase and the model constructed to solve a problem. It was the problem model which provided recall of earlier problems and not a reproduction of the textbase. Learning, according to Kintsch, depended

---

[2]*Nathan, Kintsch & Young (1992) explicitly distinguish between the situation model and the problem model. In Kintsch's earlier formulations (e.g. Kintsch, 1986), the problem model is described as a type of situation model.*

on the problem model constructed from examples, and remembering depended on the coherence of the text. For example, common terms repeated in succeeding sentences leads to greater coherence and greater recall (Kintsch & Van Dijk, 1978). He argued that it was easier (at least for children) to form an appropriate problem model if there is a concrete, familiar structure. However, other studies (e.g. Cheng & Holyoak, 1985; Novick, 1990) have shown that this is not the whole story. Problem solving transfer by adults and children from *abstract* representations can take place.

The distinction between a propositional representation of a text and the elaborated situation model was examined by Tardieu, Ehrlich, & Gyselinck (1992). They argued that novices and experts in a particular domain would not differ in the propositional representation they derived from a text but that there would be differences between the two groups in the situation model[3]. Tardieu et al. found that experts performed better on inference questions than novices but there was no difference between the groups on their ability to paraphrase the text (i.e. they both generated much the same *textbase*).

These hierarchical forms of representation have two implications for how novices understand textbook explanations and examples. First, since they are unfamiliar with the domain, they tend to have only a propositional representation of the surface features of the examples. Using examples to solve further problems means matching propositions and is unlikely to be guided by an understanding of the deeper relational structure. Second, novices may not know enough to make the necessary inferences. It follows from this that expository texts should remove the need for novices to do so, and the explanations they provide should be as comprehensive as possible.

3.2. Other definitions of problem representation

Before looking through the textbook for an example to help her, the subject had already categorized the problem as an example of recursion. This is the abstract principle underlying the solution. When she first categorized the problem as such, she did not seem to know how to convert it immediately into a working program. Her representation of the problem was, therefore, incomplete and fragmentary. She was able to relate aspects of the problem to the more abstract concept of recursion, but could not relate that to the form the solution should take.

---

[3]*Here the situation model and the problem model are synonymous.*

The failure of students to generate an adequate problem representation is well documented. Glaser (1984) provides a definition of problem representation which begins: 'We define a problem representation as a cognitive structure corresponding to a problem that is constructed by a solver on the basis of domain-related knowledge and its organization.' What this definition emphasizes is that a problem is a construct. It is not something in the outside world but something inside someone's head. He continues:

> At the initial stage of problem analysis, the problem solver attempts to 'understand' the problem by construing an initial problem representation. The quality, completeness, and coherence of this internal representation determine the efficiency and accuracy of further thinking. And these characteristics of the problem representation are determined by the knowledge available to the problem solver and the way the knowledge is organized.

Since novices are unlikely to have a complete problem model of a complex problem, their representations will be impoverished in some way. Their problem model is likely to be fragmentary and fuzzy and the elements of it are unlikely to be integrated into a structural whole, so variations in its 'quality, completeness and coherence' will have a distinct effect on problem solving.

Other researchers have also described the quality that early representations are likely to have. Barsalou (1989) states that the information brought to bear by a student attempting to solve an exercise problem will include 'abstracted properties of earlier problem solving episodes, exemplars, fragments of exemplars, and fragments of intuitive theories'. Ross (1984) also maintains that problem solving is governed by information constructed from elements of the current problem which remind the solver of earlier similar problems which in turn help the solver reconstrue the current one. Chi, et al. (1989) and Nathan, et al. (1992) argue that individuals differ in the 'completeness' with which they encode problem instructions and that this gives rise to individual differences in skill learning. Problem representation is therefore not a straightforward all-or-nothing process but a dynamic, reconstructive, and possibly haphazard affair.

Such fragmentary representations are not conducive to a concept-driven problem solving process. To solve a problem by analogy requires the solver to recognize the underlying similarity between two superficially dissimilar problems. It involves a one-to-one systematic mapping of a hierarchical system of relations that holds in one domain onto another (Gentner, 1989; Holland, Holyoak, Nisbett, & Thagard, 1986; Holyoak & Thagard, 1989a; 1989b). For this to happen, the solver must already have an adequate representation of the earlier problem that incorporates an understanding of a problem's

underlying relational structure. The evidence for the kinds of representations novices can generate suggests that they lack this type of knowledge.

3.3. Types of knowledge representation: Declarative and procedural knowledge

Solving a problem not only depends on the situation model derived from a text, but also on a solver's pre-existing knowledge base. For a successful solution (when no example is provided), the representation of the problem has to include domain-relevant knowledge. From their reading of expository texts and their previous problem solving experience, students will have learned some new concepts and learned to do certain things within the domain. For instance, the subject gave some indication that she understood the principle of recursion. What she was less sure of was how to incorporate what she knew of it into a working SOLO program. She had had practice at using NOTE and CHECK procedures and could use them in relevant contexts (see Appendix 3). She therefore knew some facts and concepts in SOLO, and knew some procedures for manipulating the SOLO database. These kinds of knowledge are often referred to as declarative knowledge, or *knowing that*, and procedural knowledge, or *knowing how*.

Declarative knowledge can be of three types (Ohlsson & Rees, 1991):

a) Factual knowledge consisting of assertions about specific objects or events such as 'Rome is the capital of Italy'.

b) Episodic knowledge, in fact a subset of (a), consisting of assertions about a specific spatio-temporal context: 'I saw Jim yesterday'.

c) General principles or abstract knowledge consisting of assertions about universals. Principles are propositions with quantifiable variables. The following arithmetic principle can be applied to an infinite number of cases:

A set of numbers always yields the same sum, regardless of the order in which they are added.

Recursion is another example of this type of declarative knowledge.

Procedural knowledge specifies the actions that are to be taken under certain conditions. It often refers to the ability to perform skilled actions. The distinction between declarative and procedural knowledge can be understood in terms of someone learning for the first time how to drive a car. The learner driver may have a great deal of declarative knowledge about the sequences of actions involved when changing gear, but this does not mean that he or she will be able to execute that sequence smoothly when called upon to drive a car for the first time. Procedural knowledge comes about through *practice* at performing the actions of driving.

In many theories of problem solving and skill acquisition declarative knowledge is a necessary precursor to procedural knowledge (Anderson, 1983; Byrnes, 1992).

More formal distinctions can be made between them in terms of goals. The semantic and episodic knowledge in declarative memory is goal-independent in that it can be used in a variety of circumstances. *Procedural knowledge, on the other hand, can only be understood* in terms of the goals one is trying to achieve. It involves the knowledge of what to do given a certain set of circumstances. Furthermore, declarative knowledge can be true or false: 'We are coming out of the recession', 'Italy is the capital of the Ukraine', 'Lloyd-George knew my father'; procedural knowledge is simply more or less effective in achieving one's goals, such as:

IF the goal is to get to the city centre by 5.00 p.m.
THEN look up the bus timetable;
walk to the bus stop;
get on the bus;
buy a ticket;
etc.

The goal of teaching is to equip learners with new conceptual knowledge and a new set of procedures for action incorporating that conceptual knowledge. The richer the set of relations that link concepts, the 'deeper' will be the learners' understanding of these concepts. Procedures, too, involve understanding. One can follow a set of instructions, as the ordinary soldiers did on D-Day, while being unaware of the overall plan; or, like the generals, one can have a privileged view of the meaning of the procedures - what they will accomplish and how they will accomplish it.

3.4. Understanding concepts

In the scenario in part 1, the subject 'understood' that the problem was an example of recursion involving a chain relation:

ATTIC OVER BEDROOM1 OVER LANDING OVER CUPBOARD etc

Despite this she could not immediately write a procedure to solve the problem. Instead she accessed an earlier example and copied it altering the values to those in the target problem. She knew that the two examples were in some way similar. Had she typed in this program and then 'FLOOD ATTIC' she would have found that the answer was correct, and thereby given the impression that she understood recursion. As it was, she wrote the solution by hand and then indicated that she was unsure if the solution would go through the whole database. *In what ways can her 'understanding' be characterized?*

Definitions of understanding abound. All of them emphasize the notion of 'flexibility'. Understanding implies the ability to *adapt* a conceptual representation to a novel situation. Eylon & Reif (1984) regard understanding as 'the processing of the original problem presentation to construct a meaningful internal representation that can be manipulated by the solver in order to produce the desired result.' Hiebert & LeFevre (1986) use the term to describe 'the state of knowledge when new [...] information is connected appropriately to existing knowledge' (which they acknowledge is more like Piaget's notion of *assimilation*). Mayer (1989) defines it as: 'the ability to use learned information in problem solving tasks that are different from what was explicitly taught', which in turn is related to Wertheimer's (1959) distinction between understanding and rote learning. Wertheimer referred to understanding as the 'meaningful apprehension of relations'; a form of understanding which is useful in new situations. Rote learning is where a learner follows a memorized procedure. Skemp (1978) also distinguishes between 'instrumental' understanding - the rote performance of procedures - and 'relational' understanding - the performance of procedures with 'understanding'.

The subject had not yet got to the level where her 'apprehension of relations' was meaningful. How then could she reach this level? Newell & Simon (1972) suggest a 4-part theory of how a procedure or algorithm is acquired:

1) The algorithm is followed by step-by-step reference to a 'recipe' stored in 'external memory'. The 'external memory' may be a textbook explanation of a procedure, such as an equation for physics problems, a rule written on the blackboard, or the INFECT procedure in the SOLO training manual. However the rule, procedure or algorithm is presented, the learner has to look back to it in order to apply it to a current problem since it is not yet memorized.

2) The 'recipe' is memorized (stored internally) but still has to be executed by step-by-step interpretation. The learner now has the necessary declarative knowledge of the procedure, and problem solving involves *interpreting* this declarative knowledge (cf. Anderson, 1983).

3) The memorized recipe is mechanized and is therefore executed directly and without interpretation. That is, the declarative knowledge has become proceduralized and compiled. Problem solving now involves executing the relevant procedures.

4) Independently of the previous sequence, understanding is acquired of the logical justification of the algorithm - of why it works. With experience of examples such as INFECT and FLOOD, the learner comes to understand why the form of these procedures in SOLO reflects the underlying principle of recursion. Another way of putting this is to say that the learner develops a theory of *causation* in a particular domain through a (domain-general) understanding of *causality* (Pazzani, 1991).

Understanding, according to this model, appears as a by-product of the procedure being memorized and mechanized. Problem solving can still take place before the 'recipe' is memorized and before the underlying structure is understood. Since the subject does not know the procedure for simple recursion problems, she is still at step 1.

3.4.1. The black box and the glass box

Conceptual knowledge often refers to a 'knowledge rich in relationships' (Hiebert, 1986). The richer the web of relationships, the better the understanding that derives from it; that is, the better able the person is to adapt that knowledge to new situations. Procedural knowledge need not access conceptual knowledge. Procedures can be learned by rote and applied to similar new situations. In this case they will fail if the situation contains features which differ from the context in which the procedure was learned. For example, someone with little knowledge of cooking can nevertheless succeed in producing something edible by following a recipe. However, if a recipe requires a particular spice and that spice jar is empty, then the procedure will fail since the novice cook is unlikely to know what other spice might do in its place. A cook with a richer knowledge of spices would be able to adapt the recipe by substituting a different spice for the missing one.

The distinction between understanding the function of a procedure and merely copying it has been characterized as the 'black box versus glass box' distinction (DuBoulay, O'Shea, & Monk, 1989). For example, someone can easily drive a car without knowing what is going on under the bonnet but if something goes wrong with the car then the person who has some knowledge of internal combustion engines will be more likely to be able to fix it and carry on the journey than the driver who lacks this knowledge. To the driver who has an understanding of the interrelation between the parts of a car, the operation of the car is therefore 'transparent'. To the driver who has no such understanding the car's operation is 'opaque'.

Not everyone accepts that there is a rigid distinction between the declarative and procedural knowledge (Laird, Newell & Rosenbloom, 1987; Newell, 1990; Silver, 1986), especially with regard to conceptual and procedural understanding. Using the same examples, Silver (1986) points out that procedural fluency (driving a car) does not have to rest on conceptual fluency (e.g. knowledge of internal combustion). He argues that conceptual knowledge is neither necessary nor sufficient for procedural knowledge but rather that both are interrelated.

3.4.2. Procedural and Teleological understanding

A learner who has reached stage 3 in Newell & Simon's list and who possesses procedural understanding has learned a set of rules for achieving a goal or subgoal. Applying the procedure accurately to a novel problem will normally lead to a correct solution. Nevertheless, lacking the knowledge of the design behind the procedure, a solver is unlikely to be able to adapt it to unusual problems, and may also be prey to the effects of negative transfer (Catrambone & Holyoak, 1989; Kessler & Anderson, 1989).

Teleological understanding is one where the learner understands not only a correct procedure but also the design behind that procedure. This form of understanding is that possessed by experts. If the design behind a procedure is known then learners can use it to reconstruct parts they may have forgotten. It allows a learner to adapt procedures to fit novel problems and can be used as the basis for intelligent analogies (Van Lehn, 1990). This form of understanding can be seen as the ideal for which a learner is striving.

It is because of the flexible nature of teleological understanding that many researchers believe that teaching is more effective when the explanations of novel constructs contain an explanation of the underlying structure. In other words procedural knowledge should be rooted in conceptual knowledge (Hiebert, 1986; Byrnes, 1992). This is the same argument advanced for the inclusion of schemas (Eylon & Reif, 1984; Smith & Goodman, 1984) or system models (Robertson, 1990) in instruction.

3.4.3. A taxonomy of procedural types

Two of the phases that the subject in the SOLO problem-solving exercise engaged in were *search* and *imitation*. The first involved a search for information she lacked. This type of search can either be random (solvers might search through the book for anything that looks like it might help) or focused (solvers may look for examples of a concept which they have incompletely understood). In the imitation phase they apply a procedure, which is embodied in an example, whether they know what the underlying procedure is or not.

VanLehn (1990) provides a taxonomy of procedural knowledge types.

1) *Unprogrammed behaviour;* which includes heuristic search such as trial and error, means-ends analysis, generating and testing hypotheses;

2) *Programmed behaviour;* which can be further divided into subtypes:

a) the learner knows a procedure and understands it,

b) the learner knows a procedure but must follow a written recipe, and

c) the learner knows a procedure but does not understand it. VanLehn argues that knowing a procedure but not understanding it is the norm in his various studies into children's learning and errors (Brown & VanLehn, 1980; VanLehn, 1982; VanLehn, 1986; VanLehn, 1989).

The search the subject engaged in is akin to VanLehn's unprogrammed behaviour. The subject searched through the training manual because she was unsure what to do next - that is, she could not resort to programmed behaviour. When her search was successful she switched to some form of programmed behaviour, where, having found a relevant procedure (the INFECT program) she followed the recipe it embodied.

The notion of imitation incorporates both unprogrammed behaviour and the programmed behaviour outlined in 2b and 2c. The subject in chapter 1 and the children in some of the studies by VanLehn had not yet learned a procedure by rote. In many of the cases studied in the laboratory only one or two example problems are given which illustrate a procedure. The subject has to map the features of the source problem onto the target and apply the procedure as it is instantiated in the source problem by extrapolating it to the target. When the current problem is a distant variant of an earlier example, the solver tries to follow the recipe in external memory (the example solution), but reaches an 'impasse' when the recipe has to be modified in any way. At this point the solver has to fall back on unprogrammed behaviour. We cannot therefore assume that the learner knows or understands the procedure instantiated in the example well enough to extrapolate it to the target.

3.5. Summary - Which comes first: Procedural or Declarative knowledge?

There are several reasons why novices may fail to make elaborative inferences from a reading of a problem. Firstly, their representations of the text are often fragmentary and incomplete, since they may not know what aspects of the text are important or relevant to the solution. Secondly, they require practice at solving problems before they develop the necessary inference rules. In other words their declarative knowledge of the domain (or conceptual knowledge) is not in a form that can support inferences.

Many models of learning such as ACT*, SOAR, Sierra, PI, etc., emphasize the deriving of knowledge from experience. That is, procedures are created from experience of instances of problem solving. During this phase learners might show procedural skill but little conceptual understanding. An example would be learning to write recursive functions in LISP from a purely 'syntactic' point of view (Hasemer & Domingue, 1989); one can learn to write recursive functions successfully without at first a full understanding of how recursion

works. Even when concepts are not fully understood they can nevertheless be 'powerful thinking tools' (Boden, 1972). Procedures in mathematics still allow problems to be solved even though the problem or procedure is not fully understood. VanLehn (1990), amongst others, has shown that having a procedure which successfully solves a problem involving a concept does not mean that the student has a proper understanding of the concept (Brown & Burton, 1978; Brown & VanLehn, 1980; VanLehn, 1986; VanLehn, 1990; Young & O'Shea, 1981).

The alternative is learning a procedure from a principled understanding of a concept. For example, it is possible to have a good understanding of the concept of recursion without knowing how it is instantiated in a SOLO function. So even though a concept is understood by a student this does not mean that the student has acquired the necessary procedures to implement that concept (Ohlsson & Rees, 1991). This was the case with the subject who, having found the first recursive INFECT procedure, interpreted it in the light of her understanding of recursion.

Byrnes (1992) proposes a 'dynamic-interaction' model to account for the interface between conceptual and procedural understanding. His model assumes that concepts and procedures are stored in separate memory systems. The interface between the two types of knowledge involves the activation of procedures that are indexed to concepts. The procedures, however, do not define concepts. Silver (1986), in contrast, believes that the two are more fundamentally interrelated and that there is little point in distinguishing between them. Just as there is a conceptual basis for procedures there is also a procedural basis for concepts. For example, the concept 'equilateral triangle' may include procedures for distinguishing between examples and non-examples of such triangles. Concepts can therefore be defined by the operations that apply to them.

If conceptual knowledge is 'rich in relationships' then concepts are, by definition, learned with meaning. Procedures, on the other hand, may or may not be learned with meaning, according to Newell and Simon. This leads to a paradox. If conceptual understanding comes first then procedures are surely also learned with meaning, since those procedures make reference to known concepts. If, however, procedures are acquired first then conceptual understanding comes *after* the procedures have been learned. In other words it is possible for procedural knowledge to *precede* declarative knowledge.

If solvers imitate a procedure illustrated in an example, then their understanding of the procedure and the concepts embodied in it can be generated *as a consequence* of the process of using that procedure.

4. The problem of access

Accessing an earlier example involves finding a match between whatever representation the solver has of the target and a potential source. In APS accounts, the source is presumed to be in LTM. Retrieving an analogue occurs through some form of hint, or through a match between the problems' surface features or structural features. In the IPS account, the subject can be reminded of a potential source through some form of hint, by being presented with a relevant analogue, or after a search through training material. Nevertheless, the solver still has to recognize some kind of similarity between the target and any potential source. This section deals with problem similarity and what is necessary for successful retrieval.

4.1. Surface and structural similarity

Problem similarity depends on two aspects of problems: those features of a problem category which can vary - the surface features - and those features which are invariant - the structural features. Changes in the surface features do not affect the solution procedure, whereas changing the structural features does. These features have been shown to have differential effects on the access and use of analogues.

The strong view of analogical access involves accessing an analogue through some similarity in their underlying structure. A 'weaker' view of analogical access involves being reminded of a plausibly relevant analogue through some other kind of similarity, usually a superficial one. Remindings can occur if those similarities are salient (Holyoak and Koh, 1987), or 'transparent' - that is, if they are similar kinds of things (Gentner & Toupin, 1986). The salience depends on the solver being able to infer that particular surface features are likely to provide a link to some underlying causal relationship which holds in both the source and target.

A weaker form still involves accessing a source problem by being given a hint to use one, or by finding an earlier problem 'by chance' while looking through a training manual. Retrieving a potential source can also come about through a variety of reasons, none of which has anything to do with recognizing similarities in either structural or surface features. An example may be chosen because it is in the same section of the textbook as the exercise problem. The subject in the protocol categorized the problem as recursive without reference to the manual and then found, by searching through it, a source that was also an example of recursion.

4.2. Recall of a relevant source

After reading and trying to understand the problem, the subject in Chapter 1 searched through the manual for something that would help her solve it. When solvers have formed an initial description or mental model of a target problem, they must either a) be able to access a plausibly relevant representation of some earlier problem from memory, or b) match the description with one already presented. The subject was unable to solve the current problem immediately, so her representation of the target problem was incomplete; that is, she was not aware of its underlying structure. She was aware that it was a recursive problem, but that representation was at too abstract a level for her to solve the problem - she could not remember how the principle was instantiated in a procedure. When solvers have an incomplete representation of a problem, they cannot access an earlier one on the basis of structural similarities. Instead, superficial similarities provide the main cues to retrieval of a plausibly relevant source analogue (Brown & Kane, 1988; Ferguson-Hessler & Jong, 1990; Gentner & Toupin, 1986; Holyoak, Junn, & Billman, 1984; Holyoak & Koh, 1987; Novick, 1988; Stein, Way, Benningfield, & Hedgecough, 1986). The subject in the SOLO experiment found an example problem, not because of either its surface or its structural similarity, but because it was the first example of recursion she happened to come across in the textbook.

4.2.1. Memory for earlier problems: Norman & Bobrow (1979)

The reasons why superficial features of a problem are important in early problem solving have to do, in part, with how information is encoded and stored in memory. In studies of recall, the likelihood that information learned earlier will be accessed increases as the similarity or degree of overlap between it and the information available at the time of the test. The same processes operate in problem-solving where the likelihood of accessing a relevant source analogue increases as the similarity between the encoding of the source problem increases (Barclay, Bransford, Franks, McCarrell, & Nitsch, 1974; Tulving & Thomson, 1973; Holyoak and Koh, 1987). Accessing an earlier similar problem therefore depends on the representation or *description* that is derived from the current one, as well as how the earlier problem is encoded (Tulving & Thomson, 1973). The description one forms of an entity, according to Norman & Bobrow (1979), is defined as 'a collection of *perspectives*, each of which is a way of viewing that entity in terms of a previously known *prototype*.' Older records can serve as prototypes for new ones. The new record is an *instance* of the prototype and can inherit some of its properties. Prototypes are not the same as abstract schemas since they can be episodic in nature.

Information about goals - exactly what it is that is to be achieved - and feature information, that is: syntactic, semantic or structural information, are used to construct a specification of a problem (essentially a description of the target along with verification criteria). This information constitutes a memory probe. A source example also consists of a description of a number of episode fragments, ones which contain partial goal information as well as partial information about actions that are performed during this part of the episode. Feature information might include such things as text content, story line and possibly even irrelevant surface information such as the font in which a text is printed. Spencer & Weisberg (1986), for example, found that context heavily influenced the likelihood of transfer in the Fortress and Radiation problems and argued that whatever schema a subject had derived from the source problem must also contain context specific information.

Armed with a description derived from the current problem, the memory base is searched for the best matching description. On recall, information is sought (or descriptions are elaborated) from the environment, the nature of the retrieval specification, and the information retrieved from memory. The retrieval process, then, involves a cycle of:

a) Forming a retrieval specification; which includes a description of the target as well as verification criteria,

b) A matching process; which refers to how good the description is. If it is highly specific then this may lead to direct access of the earlier problem. Otherwise there is likely to be some kind of spreading activation to access a suitable earlier problem. Success depends partly on how well the description discriminates between other possible memory records.

c) Evaluation; retrieved information is evaluated against the verification criteria included in the retrieval specification.

For Holyoak and for Ross the process of analogical transfer is reconstructive. Ross's view of analogical transfer is outlined in his 1984 paper. Once a description of the current situation is partially matched with a description of a memory fragment, the rest of the memory fragment is 'pieced together' guided by goal, action and feature information. That information provides constraints on the order in which the episodic fragments could have occurred. Lacking the high-level knowledge of an expert the novice has only the current target problem to guide this reconstruction. The target problem fulfils two functions: it is an instantiation of the problem structure even though the learner may as yet have no knowledge of its abstract structural aspects; it is also a memory probe which is used (according to Ross & Sofka, 1986, and Ross, 1989b) to recall and guide the reconstruction of a previous example. Once a plausible source analogue is identified, the actions that were applied in the earlier task are applied to the current target task by analogy.

Similarly, Holyoak talks of the mental model of a retrieved source analogue model as being 'unpacked'. Since Holyoak views models as being hierarchically ordered, this unpacking proceeds from the top down. The mapping and transfer processes are iterative until either a new target model is constructed or the analogy breaks down.

4.2.2. Memory for analogues

In within-domain analogies one might expect that an analogue would be easier to access than, say, the Fortress problem would be when a subject is presented with the Radiation problem. There is very little surface similarity between these problems. In Gick & Holyoak's experiments, there was either an interpolated task or a delay between presentation of the source and target. In teaching texts, on the other hand, the examples and the test problems are often superficially very similar, they are usually in the same section of the textbook and so are easy to access, and often follow on immediately after one another. Nevertheless, despite the similarity of within-domain problems, students can have a great deal of difficulty in adapting an example solution to solve an exercise problem.

Reed, Dempster & Ettinger (1985) presented subjects with an algebra word problem about two cars travelling along the same road at different speeds. In one experiment, they presented one group of subjects with an algebra word problem containing an explanation of the solution and then gave them an 'equivalent' problem (a close variant with a very similar cover story) to solve, that is, the test problem was identical to the example except that some of the values were changed. A control group was shown an 'unrelated' practice problem which was entirely different from the target problem. Neither group was allowed to consult the example problem. The results showed no significant difference between the groups for success in solving the test problems. Either the subjects could not remember the earlier problem or they could not understand it.

In a second experiment, Reed et al. repeated the procedure used in the first experiment but this time the solution explanation was 'improved'. The explanation also included a table as a way of re-representing the underlying equation. This time there was a great improvement in the subjects' ability to solve an isomorphic problem from memory. This suggests that the main difficulty the subjects had in the first experiment was understanding the earlier solution. The new version of the problem, although longer, allowed subjects to understand and therefore remember it better. However, another group of subjects had been given a distant variant of the example problem. Their results were worse than those of the subjects who were given the unrelated practice problem. This suggests that they did *not* understand the original problem, since presumably they could

remember it as well as the group who were given the close variant. The fragments of the earlier problem that they could remember were not enough to allow them to reconstruct the solution plan. In fact, it led to negative transfer.

Reed et al. could not induce transfer to the distant variant despite their efforts to make the explanation of the problem more comprehensive (such as removing inferences and suggesting the use of a table as an intermediate representation of the problem). Subjects were still unable to transfer their learning on the training problem to the test problem. This is not all that surprising since Reed et al. trained their subjects to solve a problem that was different from the one they were expected to solve. Even though the underlying equation was the same in both the training and distant variant, the solution procedure was different.

It appears that, when students are given example solutions to complex problems, particularly in formal domains such as mathematics and physics, they are unlikely to remember or fully understand them. They often cannot access an example from memory, and even if they do, they may be unable to understand the underlying structure well enough to use it in the 'top-down fashion' proposed by Holyoak. Since they can, however, solve isomorphic problems, whose surface features are very similar, they must be using some method of problem solving that does not require them to understand the structure.

4.3. Problem similarity

Two problems can be termed similar for a variety of reasons. They can be similar in that they have similar *objects*. For example, both may deal with IBM salesmen choosing which mechanic to work on their car. Despite the similarity in the content of the problems, the underlying solution structure may be entirely different. The two problems may be probability problems involving different formulae for their solution. Nevertheless, solvers may deem them to be similar due to their surface features. On the other hand, even though the surface features may be entirely different, the underlying solution principle may be the same, as in Gick and Holyoak's Radiation and Fortress problems, or the INFECT and FLOOD procedures in SOLO. The former is an example of a between-domain analogy, and the latter is a within-domain analogy.

4.3.1. Justifiable and non justifiable analogies

In within-domain analogies the kind of analogizing which takes place may be different from that which occurs in between-domain analogies. The closer two analogues are to one another in terms of their superficial features, the easier it is to map those features from

one to the other. However, even if people can map the superficial features this does not mean that they can map the underlying explanatory structure.

Dejong (1989) classifies analogies into two types - justified and unjustified analogies. Justified analogies are those which allow the solver to abstract out some form of generalization about the class of problems to which the problem and its analogue belong. This abstraction or schema can thereafter be used to solve problems of the same type. For example, having been shown a solution for the Fortress problem and using it to solve the Radiation problem, solvers are supposed to derive a 'known abstraction' (what Gick and Holyoak ,1983, called a 'convergence' schema) which can be applied to later problems of the same type. Such problems are 'justified' because the mappings are 'a manifestation of an underlying common abstraction' which may be used to solve later problems.

Unjustified analogies are those which do not have a common abstraction. The example Dejong gives is a metaphor from Gentner: 'the sweet fruit of patience'. The hearer or reader must infer a *temporary* mapping between 'sweet fruit' and 'patience' that is specific to this metaphor. Anderson (1989; Anderson & Thompson, 1989) provides an example of problem solving from an earlier example where the solver has to make an analogy with the *function* of the source example. He gives two recursive LISP functions *factorial*, which calculates the product of the integers to a given number *n*, and *summorial* which calculates their sum. *Factorial* is the source used to determine the solution of the later *summorial* function in the target. Dejong argues that in this case there is no common abstraction which can be later used to solve a like problem. For that reason the analogy is 'unjustified'.

At the other end of the spectrum there are researchers who are looking at our understanding of metaphors such as 'experience is the comb nature gives us when we are bald' and the creative use of analogies (Johnson-Laird, 1989). Gentner (1989) refers to analogies as being either *in context* or *in isolation*. By an analogy in context Gentner is referring to analogical problem solving proper, where the solution structure in one domain can be extrapolated and used in another, as in Gick and Holyoak's variations of the Radiation problem. Analogies in isolation refer to analogies one finds in metaphors.

4.3.2. The relation between surface and structural similarity

Problem similarity therefore forms a continuum from near variants to distant variants. Gick & Holyoak (1983) and Holyoak & Koh (1987) provided subjects with a range of problems which varied in gradations in terms of their surface similarity and the salience of surface features to the underlying structural similarity. In the experiments by Reed and

his co-workers (e.g. Reed, Ackinclose & Voss, 1990; Reed & Bolstad, 1991; Reed, Dempster & Ettinger, 1985) the surface features of problems were often very similar but the underlying structure varied.

People naturally expect a causal relation between surface and structural similarity. The relations between them can be fairly simple. The 'redness' of a tomato (a surface feature), for example, has a causal relation with the underlying feature of 'ripeness'. This knowledge can come about either from experience with tomatoes or from an analogy with the relation between redness and ripeness in apples. If novices are presented with an example in which the same surface features occur then they might reasonably infer that the example will involve the same underlying relation and that this relation can be applied in the current problem.

Sweller (1980) defined similarity in terms of 'shared representational predicates' which are the lower-order predicates in Gentner's systematicity theory. These representational predicates are accessible surface properties. He makes the point that shared surface features are a useful heuristic for accessing earlier problems. It is reasonable that two problems that look the same on the surface also share an underlying structure. Accessing a problem through shared surface features, he argues, is a good constraint to impose on the predicates that compose mental representations.

The relation between surface and structure is also pertinent in concept formation. Medin & Ortony (1989) argue that similarity judgements are based on our *representations* of objects (and of problems) not on actual entities themselves. The descriptive properties of objects, or their surface features, are usually related to deeper, less accessible properties. It therefore makes sense to attend to the surface properties of objects since they are a good heuristic for accessing the underlying structure.

Holyoak & Koh (1987) propose that both structural features, which play a causal role in determining possible solutions, and salient surface features influence the selection of an analogue. However, this proposal cannot be easily reconciled with the results of the experiments by Reed, Ackinclose and Voss (1990). They found that subjects did not select more inclusive source problems (ones which have more subgoals than the test problem but include all those necessary for the test problem's solution) despite the fact that such problems must share more structural features with a target problem than less inclusive problems. One would expect subjects to choose the more inclusive problem, at least in problem sets where there is a close correspondence between surface and structural features; that is, where the surface features are salient. While such a selection leads to more successful problem solving, it is not the strategy which they appear to adopt. Experts, on

the other hand, were more likely to select an isomorphic problem over a less inclusive problem but not more likely to select a more inclusive over a less inclusive one.

4.3.3. Salience as a form of similarity

For a feature to be salient it has to 'stand out' in some way. Holyoak and Koh (1987) varied the correspondences between so-called 'convergence' problems such that a broken light bulb filament that requires a laser to repair it led to greater transfer to a problem where X-rays are used to destroy a tumour than to one involving armies attacking a castle because the former were seen as more salient. Salience also includes the idea of temporal processing, which means that aspects available earlier in the reading of a problem may have a greater effect on which earlier problem will be accessed. In distant variants of a problem type, there will naturally be few shared salient features and competing associations may be activated that block retrieval of remote analogues.

Vosniadou (1989) argues that salient similarity is a way of linking both surface and structural similarity since such a feature, by the fact that it is salient, provides a direct link to the underlying structure. Both surface or perceptual similarity and structural or conceptual similarity can provide a means of accessing an earlier problem. An example of access through conceptual similarity was when the subject used the concept of recursion as the basis for searching for an example.

Stein, Way, Benningfield and Hedgecough (1986) invoke the notion of 'key concepts' from work by Weisberg, DiCamillo, & Phillips (1978). In a series of experiments in which access and use of an earlier problem were distinguished, they found that there was evidence of spontaneous transfer if key concepts were embedded in statements which had a surface structure similar to that in the target problem, and those statements emphasised the 'relevant properties' of the concept. Barclay, Bransford, Franks, McCarrell, & Nitsch (1974) found that retrieval cues which were not present in the source (e.g., 'something heavy') could facilitate access to concepts (e.g., 'piano') if the acquisition (the source) context allowed learners to focus on relevant properties of the key concept (e.g., 'the man *lifted* the piano'). No access to key concepts was found if learners were caused to focus on irrelevant properties (e.g., 'the man *tuned* the piano').

Stein et al. compared a number of conditions where a clue statement was given with a problem. To find the relationship between recall of concepts and subjects' ability to access and use them in solving a target problem, a free-recall test was included in their experiments (experiment 3). The results are given in Table 2.1 below.

Mean percentage of Problems solved as a Function of Clue Recall and Clue Statement Condition

	Clue Statement			
Clue Recall	SS/RC	DS/RC	SS/IC	DS/IC
		Informed Conditions		
Yes	78.2	76.9	9.6	14.7
No	55.8	51.9	20.5	6.5
		Uninformed Conditions		
Yes	39.7	38.5	2.5	0
No	53.2	32.0	2.5	1.9

Table 2.1. Results table from Stein et al. (1986) p. 438

SS - Similar Surface Structure between clue and problem

DS - Dissimilar Surface Structure

RC - Relevant Context: the context in which the clue was presented was relevant to the problem solution

IC - Irrelevant Context: the acquisition context was irrelevant to the solution

As can be seen from the table, spontaneous transfer of information in problem solving tasks is a function of a) characteristics of the source context, whether the source context was relevant to the target or not; and b) the surface structure of the problem statement which influences the accessibility of that information in the first place. In the Informed Conditions subjects were told that a concept they had been presented with earlier would be helpful in solving the current problem. This hint was not given to the Uninformed subjects. Their findings suggest that 'the accessibility of relevant clue information during problem-solving tasks is primarily influenced by similarity in surface structure (i.e., the use of identical content words) in the clue and problem statements' (p. 439).

4.3.4. Analogy as a continuum

The problem that was given to the subject was analogous to the INFECT problem in the training manual. She failed to access it based on the surface features of the problems. She found it by categorizing the problem correctly and using the categorization to guide her search. The INFECT example as an analogy to the FLOOD problem is both 'justified' and 'in context' since both are examples of how the underlying principle of recursion can be used in a SOLO program. The variations on the colinear distance problems used by Reed, Dempster & Ettinger (1985) are also analogous according to their shared underlying

equation, and their surface elements were identical. Looking at what some researchers have written about the notion of analogy suggests that it forms a continuum from the concrete to the abstract. Anderson & Thompson (1989) give analogy a very broad definition. Gentner (e.g. 1989) is much more strict in her definition. Because of the wide applicability of the concept Dejong (1989) refers to it as the 'curse of an alluring name.'

According to Dedre Gentner, an analogy is not simply saying that one thing is like another. *'Puppies are like kittens' or 'milk is like water' are not analogies. They are in Gentner's* terms 'literally similar' since they share the same attributes, such as 'small' in the first case and 'liquid' in the second, as well as the same relations. Gentner (e.g. 1989; Falkenhainer, Forbus, & Gentner, 1989; Gentner & Toupin, 1986) and Vosniadou (1989) argue that 'real' analogies involve a causal relation. In the analogy 'puppies are to dogs as cats are to kittens,' there is a relation between 'puppies' and 'dogs' which also applies between 'kittens' and 'cats,' and this relation can readily be explained. An analogy properly so called involves mapping an explanatory structure from a base domain (puppies and dogs) to a target (kittens and cats).

Nevertheless, two problems can be literally similar and still share an explanatory structure. A problem requiring the solver to find the time taken by a car to travel a certain distance at a certain speed is literally similar to one involving finding the time taken by a bus travelling a different distance at a different speed. The explanatory structure is the equation relating the elements of the problem statements and the solutions. The same equation is used in both problems.

Vosniadou (1989) argues that 'analogical reasoning can be employed between items that belong anywhere in the continuum from literal similarity to non literal similarity' (p. 415). At the concrete end there are examples of within-domain analogies involving two problems which are highly similar in both their surface features and in their underlying structure. For example, Anderson, Conrad, & Corbett (1989) claim that people solve the problem in LISP of finding the first element in a list, such as **(a b c)**, *by analogy* to an example which shows how to find the first element of the list **(p q r s)**. The solution to the source problems was **(car '(p q r s))** and so the solution to the target problem also uses **car** to produce **(car '(a b c))**. Dejong (1989) and Gentner (1989) have doubted whether this can be called an analogy at all. The problem can be solved by mapping the surface features and does not require that the solver is aware of the explanatory structure.

4.3.5. Gentner's structure-mapping theory

When an example is being used as an analogy, the objects (the salient surface features) in one domain are assumed to be 'put in correspondence' with the objects in another to obtain the 'maximum structural match.' Analogizing, according to Gentner (1983), involves mapping a relational structure that holds in one domain onto another. This is known as the principle of *systematicity*, which states that people prefer to map hierarchical systems of relations in which the higher-order relations constrain the lower-order ones. The structure-mapping theory is implemented as the Structure Mapping Engine (Falkenhainer, Forbus & Gentner, 1989) and uses a predicate calculus of various orders to represent the structure of an analogy.

At the lowest order, order 0, are the objects of the analogy, such as 'army', 'fortress', 'roads', 'general.' A predicate has the order 1 plus the maximum of the order of its arguments. So, where *x* and *y* are objects, GREATER THAN (*x*, *y*) would be first order and CAUSE [GREATER THAN (*x*, *y*), BREAK (*y*)] would be second order. CAUSE, IMPLIES, and DEPENDS ON are typical higher-order relations. 'On this definition, the order of an item indicates the depth of structure below it. Arguments with many layers of justifications will give rise to representation structures of higher order' (Gentner, 1989, p. 208).

Mapping an explanatory structure allows one to make inferences in the new domain or problem, since the relations which apply in the source can be applied in the target. A structure such as: CAUSE [STRIKE (ORANGE, TREE), FLATTEN (ORANGE)] can be used in a Tom and Jerry cartoon to decide what happens when Tom smashes into a tree - CAUSE [STRIKE (TOM, TREE), ?] - to generate the inference CAUSE [STRIKE (TOM, TREE), FLATTEN (TOM)].

Gentner (Falkenhainer, et al., 1989; Gentner, 1989) provides a taxonomy of similarities between problems in which analogy is distinct from other types of similarity. For her, analogizing involves a one-to-one systematic mapping of the structure of the base domain onto the target. The surface features - the object descriptions - are not mapped onto the target since they play no role in the relational structure of the analogy. The problems dealt with by Gick & Holyoak (1980; 1983) are therefore analogous under this definition.

Figure 2.1 shows different types of similarity and where they might lie on a continuum.

Literal similarity. Literal similarity is almost always found in within-domain comparisons. When problems are literally similar then not only is the hierarchical

relational structure mapped over, but the object descriptions are mapped as well. In the example 'milk is like water' many of the attributes of water can also be applied to milk. There is a fuzzy boundary between what is technically analogy with its normally exclusively shared relational structure, and literal similarity which also includes object descriptions.

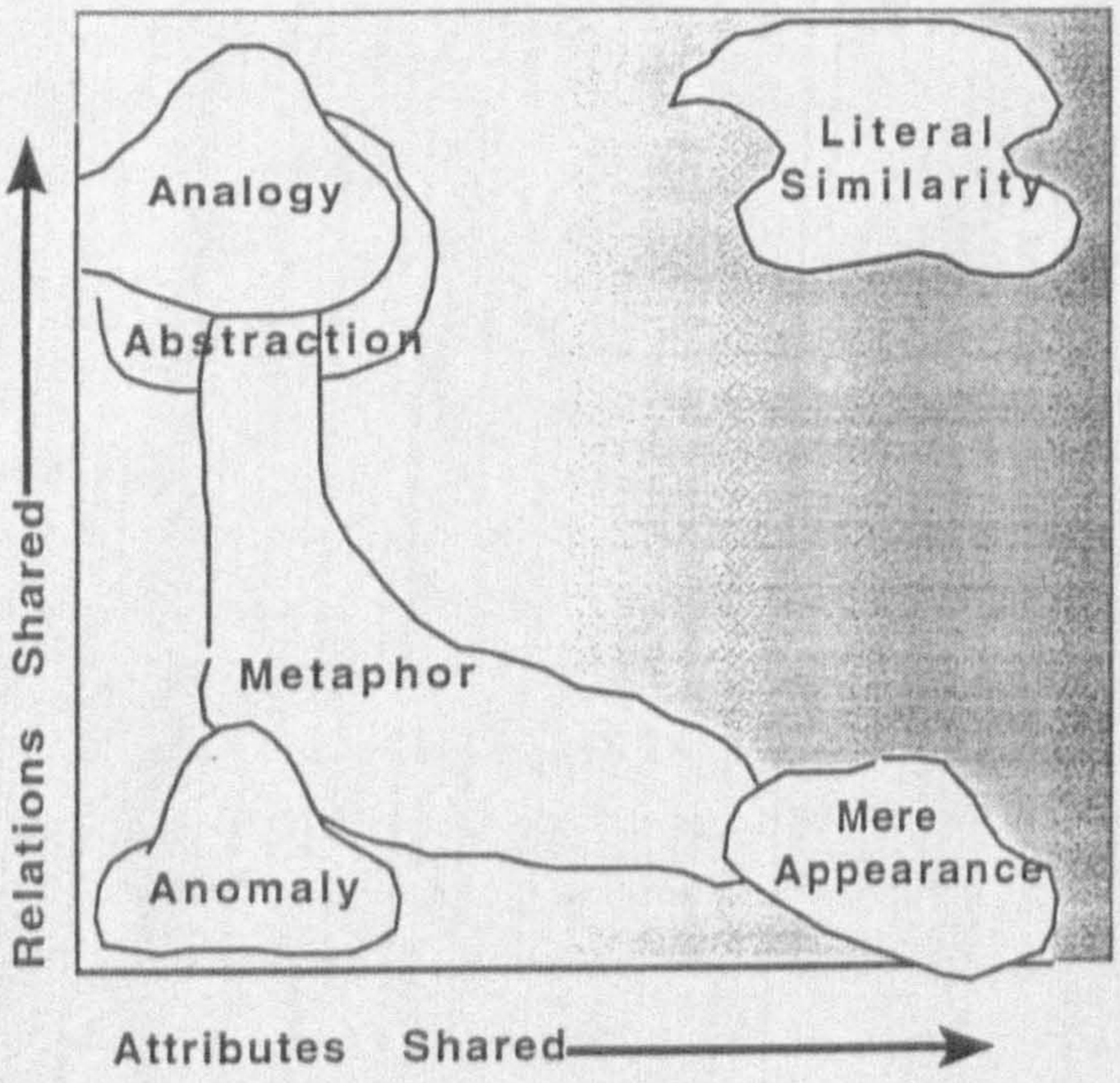


Figure 2.1. Similarity space: classes of similarity based on the kinds of predicates shared. From Gentner, 1989, p. 207.

Mere-appearance match. With mere appearance match only the lower order predicates match. The relational structure is ignored. Gentner gives the example: 'the glass tabletop gleamed like water' in which only the physical description is shared between the base and target. She claims that novices are prone to mere appearance matches.

Abstraction mapping. Where the base contains elements which are *variables* or abstract principles then mapping involves applying a known abstraction to the target as in 'heat is a through variable'. Abstraction mapping assumes prior knowledge on the part of the reader since, as in the above example, the abstraction can be used to categorize a problem or instantiate a procedure or rule. There are no concrete properties of objects in the base domain. Abstraction mapping also forms a continuum with analogy. As the object nodes in the base domain move towards the abstract variable-like end of the continuum then it becomes a relational abstraction.

The shaded area in figure 2.1 indicates where imitation fits into this scheme. Novices use mere appearance matches to access and use earlier examples. The shading includes literal similarity, in which there are shared relations as well as attributes. In such cases the earlier problem is an isomorphic within-domain analogue. That is, the two problems are 'close' variants of one another. Novices are often able to solve isomorphic problems in the same domain when only the values of certain attributes are varied; for example, when 30 mph in one problem becomes 40 mph in another.

4.4. Summary

Retrieval of a source has been found to be very difficult

a) without a hint;

b) unless the superficial features remind the solver of an earlier problem;

c) some aspect of the problem guides the solver's search for a suitable example to imitate.

The evidence suggests that surface features have a strong effect on the access of an earlier problem. Some of these features may be salient in that they relate to shared structural features. However, retrieval can be effected through irrelevant surface features such as the font used or other contextual elements.

Much work has been done on the transfer of solutions in between-domain analogies (Bassok, 1990; Carbonell, 1983; Gick, 1985; Gick & Holyoak, 1980; 1983). Gentner's studies (Collins & Gentner, 1990; Gentner & Gentner, 1983; Gentner & Landers, 1985; Gentner & Toupin, 1986), as well as those of Holyoak, have often dealt mainly with between-domain analogies, as in her studies of Rutherford's analogy between the structure of the atom and the structure of the solar system, or ahalogies with the flow of electricity. In these cases the base domain is either relatively well known or easily understood. Between-domain analogies tend to have few surface features in common. Accessing a relevant source is therefore very difficult without a hint.

In studying within-domain analogizing researchers have looked at how people retrieve and use an earlier example of the same type of problem (Anderson, Farrell & Sauers, 1984; Cooper, 1983; Green & Gilhooly, 1990b; Novick & Holyoak, 1991; Reed, 1989 ; Reed, Ackinclose & Voss, 1990; Reed & Bolstad, 1991; Reed, Dempster & Ettinger, 1985; Reed, Ernst & Banerji, 1974; Ross, 1984; 1987; 1989a; 1989b; Ross & Sofka, 1986; Sweller & Cooper, 1985). Within-domain analogies differ from between-domain analogies in that the whole domain (including both base and target problems) is new to the novice and the solution structure underlying both tends to be more complex than in the between-domain analogies that form much of the basis for APS accounts. Because of the novices' unfamiliarity with the underlying structure of example problems, recalling the relevant example is very

difficult (Reed, et al., 1985). Most studies of within-domain problem solving examine how examples are used rather than how they are retrieved. In teaching contexts, students often retrieve a useful source after a search through a textbook, or by being given a model to use.

5. The problem of mapping

Mapping, in most accounts of APS, refers both to finding correspondences between a source and a target, and analogical inference. The latter involves adapting a structured system of relations. Mapping in the IPS account is limited to mapping values that appear to fill the same roles in both problems. There are some important corollaries that follow from these views. From the APS point of view, if the solver is aware of the underlying structure of two or more analogues, adapting it to fit the new problem will not pose too many difficulties. On the other hand, if solvers are using imitation, they will find it very hard to adapt the mapping. Indeed, much of the literature on analogical transfer has demonstrated how hard transfer is to induce. To understand why transfer seems so difficult, we have to look at the roles surface (or semantic) features and structural (or syntactic) features play in adapting an analogue.

In this section, two views of mapping and adapting an analogue are contrasted. The first, the principle-cueing view, will be discussed mainly with reference to the work of Holyoak. Many others share a 'structural' view of analogy, and many AI models such as PI, ACME, ACT*, SME, PUPS and SOAR, also espouse this view of analogical mapping. The second view, the 'example-analogy' view, is represented by the work of Ross, whose views on APS are similar to the IPS account.

5.1. Mapping the surface and structural features of problems

Holyoak (1984) identifies 4 types of mapping relations:

1) *Identities;* these are elements which are the same in both analogues. These identities are equivalent to the schema that is 'implicit' in the source and might include such generalized rules as 'using a force to overcome a target'.

2) *Indeterminate correspondences;* which are those elements that the problem solver has not yet mapped.

3) *Structure-preserving differences;* These refer to the surface features of problems which, even when changed, do not affect the solution. Examples would be 'armies' and 'rays', and 'fortress' and 'tumour'. Although entirely different, such surface features do not affect the underlying solution principle.

4) *Structure-violating differences;* These differences do affect the underlying solution structure. The Fortress and Tumour problems use a solution involving 'division' and 'convergence'. However, whereas armies can be divided into smaller groups, a ray machine cannot; nor is it immediately obvious how the rays can be divided. The solution

to the tumour problem means getting hold of several machines and reducing the intensity of the rays. The operators involved in the Fortress problem have to be modified in the Radiation problem.

Structure violating differences necessitate the refinement of the selection of a source analogue. Structure-preserving differences, on the other hand, although they lead to surface dissimilarities, do not require the initial selection plan to be refined. When the solution structure is preserved despite surface dissimilarities, structure-preserving differences allow the construction of corresponding operators between target and source. Structure-violating differences prevent such operators from being applied and cause the analogy to break down.

5.1.1. The PI model of analogical problem solving

Holland et al. (1986) provide a rule-based system of inductive inference as a model for problem solving and other high level cognitive processes. A representation of a problem is a mental model made up of two rule types - diachronic and synchronic - describing changes of state and categorizations of elements in the world respectively.

Representations in PI are declarative structures called concepts which include factual information ('Lassie is a dog') as well as general rule information ('Dogs have fur') which is an example of a general concept. Whereas ACT* (Anderson, 1983) segregates declarative and procedural knowledge, PI stores both under 'concepts', so that the general properties of a concept are stored with information about individual instances. Since factual knowledge and procedural knowledge are stored together in clusters or schema-like units, PI differs also from SOAR in which all knowledge is procedural and in which there are no higher-order control structures corresponding to PI's schemas.

Analogical problem solving in PI assumes that a mental model of a source problem has to be retrieved in order for an analogous problem to be solved. Such a mental model has to contain structural information about the problem in terms of its initial state (the statement of the problem), the goal state, the operators whereby the initial state is transformed into the goal state, and the constraints which are imposed on the problem search space. This mental 'copy' of the world is termed a *homomorphism* from the mathematical structures known as morphisms in which the environment can be characterized as being in certain *states* and each state is related to its successor by a transition function.

Mental models are production rule-based representations. Input in the form of 'messages' is matched against the conditions of production rules. Messages are proposition-like

structures corresponding to the 'facts' of ACT*. They can either be information arriving through receptor systems or retrieved from declarative memory, one important example of which would be the specification of the current goal of the system. A limited number of rules may then operate in parallel and candidate rules place 'bids' to determine which will be executed, the success of which is determined in turn by the strength (a measure of the past usefulness of the rule), specificity (the more salient a rule the larger the 'bid') and support (a measure of the support from other matching messages).

Rules are activated in parallel and constitute what Holyoak & Thagard (1989a) term an 'aura' of associations. A candidate analogue is retrieved if there are multiple sources of support for it. When a threshold of summation is reached, the retrieved representation becomes available for further processing. That is, if there are many shared features between target and source, these multiple sources will summate and this summation ensures that only plausible source analogues will be activated.

According to Holland et al. (1986), mapping begins when the learner identifies the transition function for the source analogue connecting its initial state to a subsequent state. Thereby the learner identifies the actions required to produce a solution from the initial statement of the problem. However, the mapping process is also a hierarchical one in which objects in the source and target can be matched, or in which abstract categories or structures may be matched. For instance, in Holyoak and Koh's (1987) experiments some subjects were able to match objects such as 'rays' or 'laser beams' or 'sound waves' across from one problem to another. The solution is then *constructed* from the underlying schema in the source. Alternatively they may have had a *ready-made* abstract schema for convergence problems which they can map directly onto source and target.

5.1.2. Criticisms of PI

PI has been criticized from several perspectives. The first criticism by Gentner (1989) and Johnson-Laird (1989) is aimed at the notion of analogy which PI incorporates. PI was not intended to operate upon analogies other than the 'justified analogies' in problem solving. It does not deal with expository analogies. This limits its applicability and causes Gentner to wonder if Holyoak believes that there is more than one analogical process.

Holyoak & Thagard (1989) present an example of the workings of PI by applying it to the Fortress and Radiation problems. However, Dejong argues that the representation they present of the problems biases them in favour of easier mapping. Furthermore, since Holyoak espouses the principle-cueing view of analogical problem solving, then the underlying solution principle is already known. If it is already known, Dejong argues,

there is little need for analogical reasoning to be used to solve the problem. This perhaps overstates the case. Holyoak (e.g. 1985) talks of partial mappings and an iterative process of mapping and adapting an analogue. But if Holyoak wishes to argue that subjects can have a ready-made schema, then in this case there is no need to retrieve an analogue: the subjects already know how to solve the problem.

Other criticisms by Palmer (1989) are aimed at its range of applicability and its specificity. As it stands, PI is designed to deal only with relatively simple problems. If the database were to be enlarged then 'potential mappings may swamp the system'. Its production rules are also highly specific. Palmer goes on: ' Do we really need to suppose that people have a rule like "If *x* is an army and *y* is a road between *x* and some *z*, then *x* can move to *z*"?' He also points out that, since knowledge representation is so specific, problem solving would be disrupted if there were even minor changes to the rules causing them to fail. In many of the studies of problem solving in formal domains, such as the mathematics problems in those of Reed et al (1985), or in computer programming (Anderson, Farrell & Sauers, 1985), it is doubtful whether the subjects could identify the transition function, or abstract out the underlying schema from a source.

Holland et al.'s PI program represents an 'ideal' model of analogizing. It readily solves the analogous problems presented to it, and indeed, some people may solve those problems by accessing a structure in much the same way. It therefore models the type of analogizing that humans are capable of, but it may not be the most usual method that humans use. Other AI models of APS, such as those described by Anderson (Anderson, Conrad, & Corbett, 1989; Anderson, Farrell, & Sauers, 1984; Anderson & Thompson, 1989), are also powerful analogical models. Whether human beings are as powerful as the models is a moot point.

5.1.3. Mapping in imitative problem solving

The degree of manipulation of a problem representation solved through imitation is limited to assigning values to variable roles. These variable roles are established by comparing problems. The comparison process identifies perceptually similar and salient features such as the rates of travel in the experiments of Reed, et al. (1985), or the object correspondences in those of Gentner & Toupin (1986). To solve a problem by imitation the solver makes two inferences. The first is that the perceptually similar features occupy the same *role* in both problems. The second inference arises from the successful assignment of different values to the same role. If two values can occupy the same role they can therefore be variabilized.

It is rare that a source problem will have elements which are identical to those in a target. The solver therefore needs some way of establishing that the values given in a source can map onto those in a target. Reed & Bolstad (1991) state that 'if the values of matching concepts have the same structure, then this structure is copied for the test problem.' What this means is that, if, say, an algebra problem involving times and rates of vehicles were used as a source, then a structure such as 30 mph, would be copied into the target where the problem statement might give the rate of a vehicle as 40 mph with the 40 replacing the 30 in the earlier example. These 'surface structures' occupy the same role in the underlying solution structure of the two problems. They are highly salient in that they suggest a causal link to the underlying solution structure. 'Copy' in the sense used by Reed and Bolstad would be the same as imitation. We cannot, however, assume that solvers have any understanding of the concepts involved or the relation between them if they are simply imitating an example.

5.2. Constraints on the access and use of analogues

Gentner's systematicity principle assumes that the causal relations between a problem's elements constrain a solution. Causal constraints are important only if the solver understands or is aware of those causal relationships. In complex problem solving in a new domain such relationships are not well understood. It takes some experience with the new domain to develop knowledge of the causal relations within it. They can be learned because we apply our general knowledge of causality to the new domain (Pazzani, 1991). Indeed, even infants are 'predisposed to seek causal explanations' (Brown, 1989).

5.2.1. Semantic and structural constraints

Structural accounts of analogy demonstrate how syntactic constraints can operate to produce analogical transfer. Semantic constraints also affect transfer and come about through similarities in the surface features. The objects in problems can vary in the degree to which they are alike. 'Laser beams,' 'X-rays' and 'sound waves' can be seen as very similar types of things and therefore have 'high transparency.' 'Armies' and 'rays' are dissimilar objects, and have low transparency. Accessing a previous problem has been found to depend strongly on the amount of semantic similarity between objects in different problems (Gentner & Toupin, 1986; Holyoak & Koh, 1987; Ross, 1987).

Holyoak and Koh also emphasized the differential effects of semantic and syntactic similarity on retrieval and mapping. They varied the correspondences between so-called 'convergence' problems such that a broken light bulb filament that requires a laser to repair it led to greater transfer to X-rays used to destroy a tumour than armies attacking a

castle because they were seen as more semantically similar. Semantic similarity, they claim, is important for retrieval of a candidate analogue whereas syntactic similarity is more important for mapping.

Gentner and Toupin (1986) presented children with stories in which the transparency of object correspondences and the systematicity of the original story were varied. Transparency was varied by changing the characters that appeared in the stories or by changing their roles in the story (leading to 'cross-mapping'). Systematicity was varied by adding a sentence to the beginning to provide a setting and a final sentence to the story in the form of a moral summary of the tale (the systematic condition). The non-systematic condition had neither a moral nor a setting.

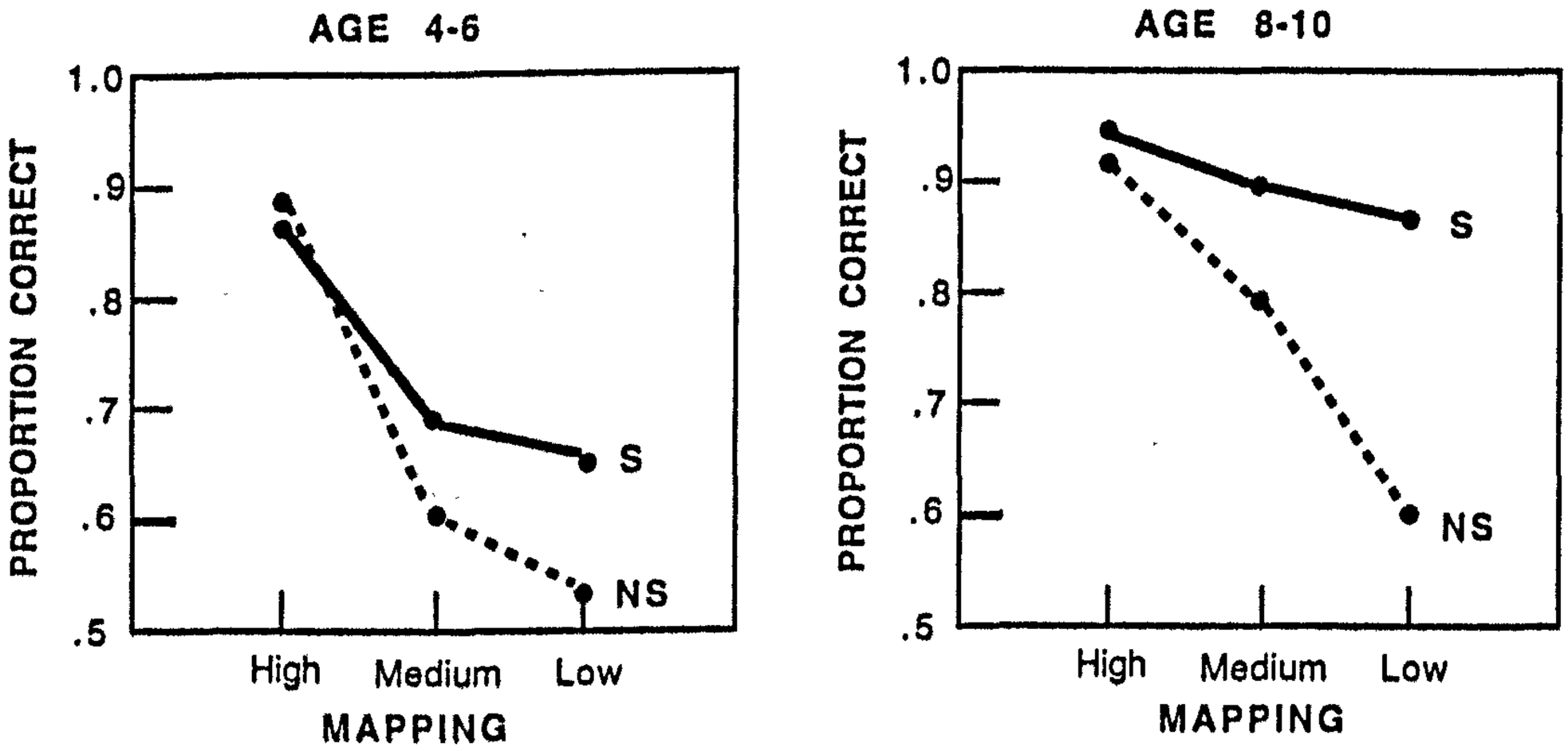


Figure 2.2. The proportion of correct responses given systematic (S) or nonsystematic (NS) stories across mappings varying from High to Low transparency. From Gentner and Toupin (1986).

The children were given certain roles to play and asked to act out a story similar to the one they had just heard. Both systematicity and the transparency of the object mappings were important factors in analogical transfer (figure 2.2). Younger children, however, relied on the transparency of the objects whereas the older ones were more affected by the systematicity of the base domain. Gentner and Toupin argue that the shared system of relations (the systematicity) between the problems helps guide the mappings of the lower-order relations.

Another way of looking at these results is to take the view that the children were attempting to imitate the previous story. Where they *understood* the story's underlying rationale they were able to adapt their characters to fit the story's structure. The rest of the time they simply imitated the sequence of actions taken by the semantically similar

counterparts in the earlier story. Again we find that analogies can be applied only when the subjects have a complete understanding of the underlying relational structure. Most of the younger children simply did not have an adequate understanding of the earlier story to be able to apply the rationale behind it.

These results do not confine themselves to children. Gentner and Schumacher (1987; Schumacher & Gentner, 1988) found the same results with adults in a different domain. Their subjects had to learn a procedure for operating a computer simulated device and then use it to learn a new device. Once again the systematicity and transparency were manipulated. The systematicity was varied by either providing a causal model of the device or simply a set of operating procedures. The transparency referred to the type of device components. The results showed that systematicity constrained learning and transfer to the target device. Transparency also had strong effects on transfer. The speed of learning the new device was greater when corresponding pairs of components were similar than when they were dissimilar. The same pattern of results was found by Ross (1987; 1989a).

Bassok (1990) found transfer between highly content-specific domains in studies of transfer between physics and algebra problems. She found spontaneous transfer when the objects and variables in the base (e.g., speed and typing rate) and target could be put into correspondence due to their similarity, just as Ross (1987, 1989) had found. That is, the objects were seen as more transparent or salient. However, there was little spontaneous transfer when those variables represented different types of quantities (e.g., speed and salary) despite the fact that the problems were isomorphic. When there was correspondence between the variables and quantities, the subjects could apply learned procedures. When there was low correspondence between the quantities, the subjects often required a hint for retrieval and the mapping became 'effortful'. Just as Ross had done, Bassok found that different types of surface similarity affect both access and use of a source problem where those surface features can be used to interpret variables.

5.2.2. Making the underlying structure more explicit

Brown & Kane (1988) attempted to find out if children could transfer their learning on the basis of surface features (a perceptual basis), or on the basis of an underlying principle. They found that children could be taught to look for the relational similarity between examples then apply that to other variants of the problem type. Children were able to access relevant source problems despite intervening tasks. Brown and Kane point out that, 'where learning can be organized round a *guiding principle*, transfer is determined by the

extent that the subject is privy to that principle, through either discovery or instruction' (p. 495).

A causal model provides learners with a means of understanding the interrelations between a set of objects and predicates. The higher-order relations constrain the mappings between the lower-order relations and literal objects. Figure 2.3a represents an example problem statement (the A box) and the solution (the B box). Linking the two there is a relational structure represented by the line from A to B. If that relational structure is known, it becomes possible to apply it to a new problem in order to generate the solution.

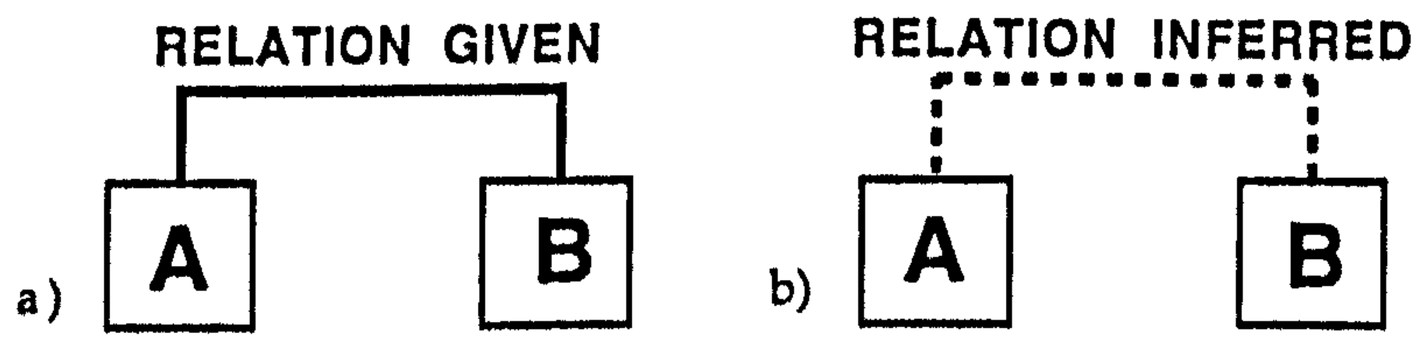


Figure 2.3. Problems in which the relation between the statement and solution is a) given and b) inferred.

If no causal model is given, as in Gentner and Schumacher's study, where a set of operating procedures was given instead, then the relation between the problem statement (the A term in figure 2.3b) and the solution has to be inferred (hence the dotted line linking A and B in figure 2.3b). By the same token the mappings between the base and target will be underconstrained. With complex problems in particular we would expect to see the mappings disrupted without a systematic causal model to guide them (e.g. Reed & Bolstad, 1991; Ross, 1987). The only resource open to the solver in such a case is to attempt to copy the actions performed in the original model and infer the mappings.

5.3. Using an analogue for principle-cueing

There are different views as to the roles that surface and structural similarity play. Ross (1984; 1987; 1989b) discusses two possible scenarios in APS: the *principle-cueing* view and the *example-analogy* view. In the principle-cueing view, learners may be reminded of an earlier example by some feature or combination of features of the current one. This reminding triggers or cues the abstract information or principle involved in the earlier problem which is relevant to the current one. This abstract or relational information can then be used to make sense of a new situation or solve a new problem with the same structure. However, principle-cueing by definition presupposes that the analogizer understands the principle in the first place. For novices studying a new subject that may not always be the case.

There are many examples of successful use of a principle presented in a 'bridging' analogy. Successful problem solving has been shown to occur when subjects are given access to an analogue which they can use. Gentner & Gentner (1983), for example, presented subjects with two analogies to the flow of electricity through resistors. In one, the analogy was with water flowing through narrow sections of pipe. In the other, electricity was likened to people passing through turnstiles in a busy station. The analogy with people passing through the turnstiles led to a better understanding of electricity flow through particular arrangements of resistors than did the narrow pipe analogy.

Issing, Hannemann & Haack (1989) performed an experiment in which they examined the effects of different types of representation on transfer. The representations were pictorial analogies of the functioning of a transistor. They presented subjects with an expository text alone, the text plus a sluice analogy, the text plus a human analogy, and finally the text plus an electronics diagram of the transistor. Unlike the human analogy in the Gentner studies, it was the sluice analogy (involving the flow of water) which led to better understanding of the function of the transistor. The human analogy was less effective and the diagram and text alone were the least effective[4].

These results can be explained by assuming that the solvers can go back to the particular expository analogy they were given and apply the principles involved to the novel situation. They also show that writers have to be very careful about which type of analogy they choose. In some circumstances, such as understanding the flow of electricity through a resistor, electricity is best conceived of as the movement of people. To understand the function of a transistor the flow of electricity is best likened to the flow of water. Variations on the particular shared relational structure or underlying principle can have a strong effect on later problem solving.

---

[4]Somewhat contrarily, Issing et al. argue that analogies depicting human-like situations are regarded as artificial and take on 'more a motivating than cognitive function.' This, they say, explains why the human analogy fails to be as effective as the flow of water. This is a bizarre conclusion given that they are taking into account Gentner's view of structure-mapping. The expository text presented to subjects talks about the 'flow of current.' People, however, don't flow; water does. So the water analogy shares more higher-order relations than the human analogy, and this would account for the stronger effect of the sluice analogy.

Most of the subjects in Gick and Holyoak's studies were at first unable to solve the Radiation problem even though they had previously been given the solution to the Fortress problem. When they were given a hint to use the earlier problem most were able to solve the new one. The hint did not include information about what to do, merely that the earlier problem would help. It was up to the subjects to work out what aspects of the source would help generate a solution in the target.

According to most models of APS, the subjects in these experiments must have had a representation of the earlier problem at a level of abstraction sufficient to be able to apply it to another problem with different surface features. In Gick & Holyoak's studies, the hint allowed their subjects to access the 'divide and converge' schema which they had generated from the solution to the Fortress problem. Their initial difficulty was accessing a suitable source problem. In the electricity flow experiments, the subject is likewise presumed to access the representation of the earlier example in LTM, extract the underlying principle, and apply it to the current problem.

Holland et al. (1986) refer to analogues as having an 'implicit' schema which is reconstructed during the solution process. In figure 2.4, *A* represents a problem statement and *B* the solution. The relation or set of relations between *A* and *B* is represented by the line linking them. If the problem is an instance of a category of problems then the solution procedure used to get from *A* to *B* can be applied to other problems of the same type. There is therefore a schema implicit in the solution. This is shown as the shaded *S* box in the figure.

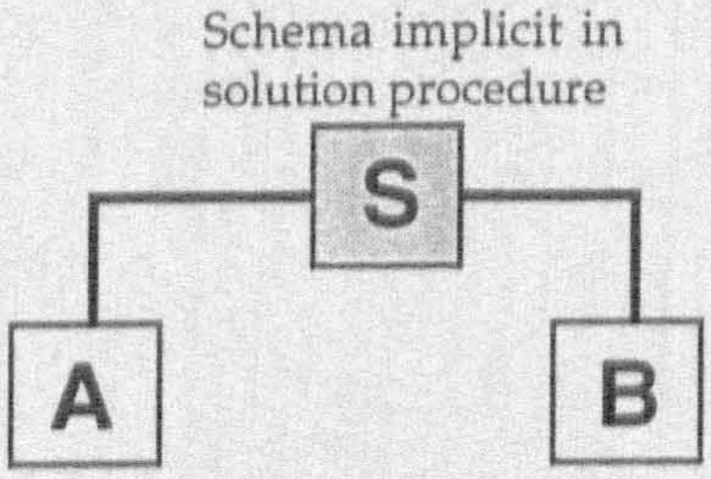


Figure 2.4. The relation between a problem and its solution involving an implicit schema.

When a source problem is accessed (*A* and *B* in figure 2.5) then the principle underlying the solution to the source is accessed (the *S* on the line linking *A* and *B*) and applied to the target (*C*) to generate the solution (*D*).

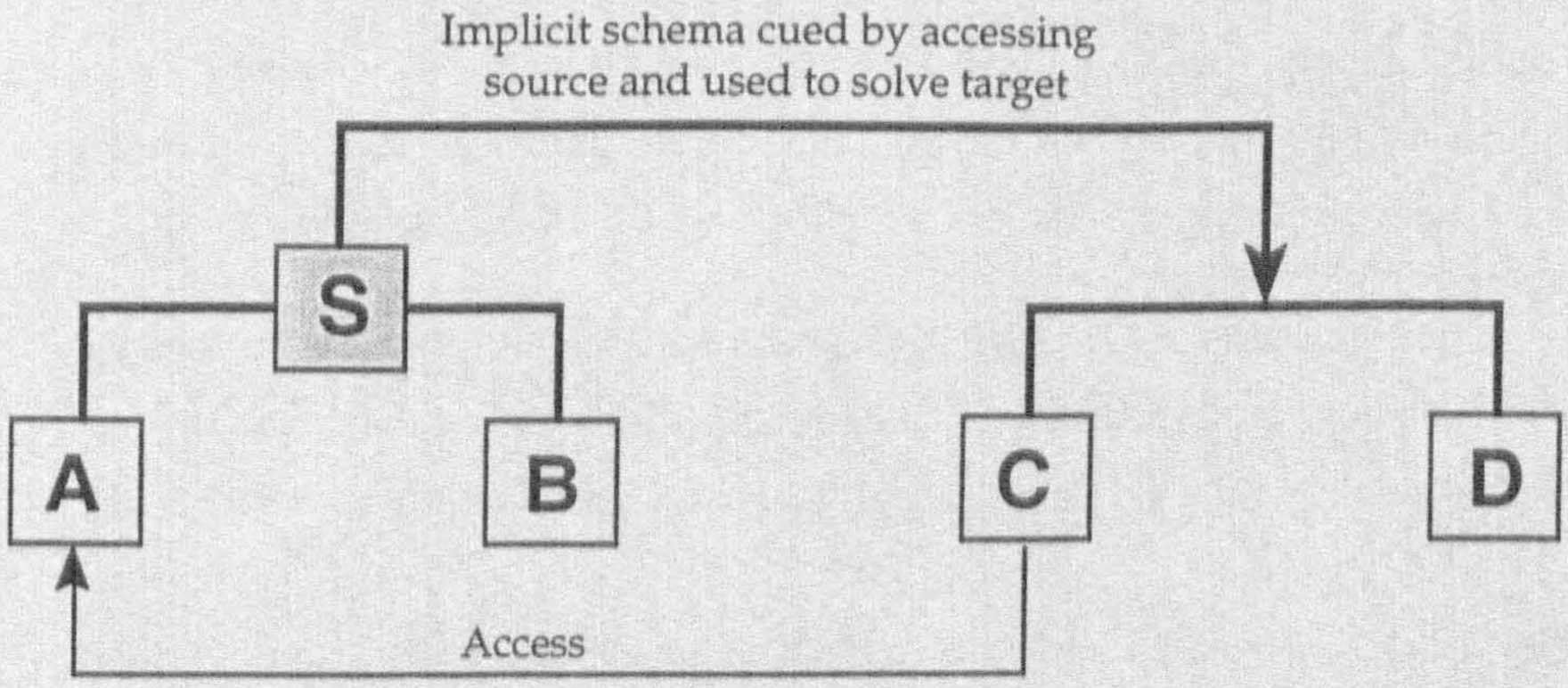


Figure 2.5. Representation of principle-cueing view of APS

In cases such as these, subjects are reminded of the analogy and the remindings serve to categorize the current problem. The abstract principle which the earlier problem or analogy exemplified is presumably already understood by the learner and the specific instantiation of the principle in the earlier problem is no longer required to solve the target. The original source problem was nevertheless important in that it allowed the learner to understand the principle in question and how it is used.

The principle-cueing view implies that the learner already has an adequate conceptualization of the solution procedure acquired from the source problem. Solving another problem from an example involves abstracting out the principle or procedure from the example and applying it to the target. This smacks of abstraction mapping except that the abstraction is 'hidden' or implicit within the example and has to be extracted before it is applied. Much of the literature on expert-novice differences has concentrated on how the correct perception of a problem can cue access to the 'problem schema' (Chi, Glaser, & Rees, 1982; Larkin, 1978). This problem schema in turn suggests a straightforward, stereotypical solution method. Novices, however, are often unable to identify the problem schema or categorize problems accordingly.

In complex within-domain analogies, it would be unwarranted to assume that the novices have a schema, implicit or otherwise, for a problem. There may be a schema implicit in the problem but there is no guarantee that it is represented in the mind of the solver.

5.4. Using an example as an analogy

The view that the role of superficial features is simply to access a previous problem has been challenged by Ross in a series of experiments. According to the example-analogy view:

> 'the principle is understood only in terms of the earlier example. That is the principle and example are bound together. Thus even if learners were given the principle or formula, they would use the details of the earlier problem in figuring out how to apply that principle to the current problem' (Ross, 1987, p. 629).

This is the case with the subject in chapter 1. She was aware of the principle but needed the example to see how the principle was instantiated within it, and then used the details - such as details of the syntax - to apply the same principle to the target.

Much of Ross' work was concerned with the effects of superficial similarities in problem access and use. In Ross (1987) the superficial similarity between example and test problems was varied in terms of the story line and the object correspondences. The correspondences between objects and variables were either similar, reversed (where the objects played different roles in the solution), or unrelated to the study problem. Table 2.2 summarizes the conditions used. The problems were probability problems with various story lines such as IBM mechanics choosing what company car to work on. In the same/same condition there were only minor superficial changes to the problem. The underlying solution structure remained the same. In the same/reversed condition it was the IBM salespeople who chose which mechanics should work on their cars. The same/unrelated condition involved computers and offices in an IBM building. The unrelated/unrelated condition involved ticket sales for a high school athletic team whose objects (teams and teachers) were unrelated to the example problem.

	Study-test relation		
Condition	Story line	Objects	Correspondence
same/same	same	same	same
same/reversed	same	same	reversed
same/unrelated	same	unrelated	unrelated
unrelated/unrelated	unrelated	unrelated	unrelated

Table 2.2. Study-test relations in Ross (1987)

Even when subjects were provided with the relevant formula, so that the task was one of instantiating the formula with the figures in the problem, the ability of the subjects to use the formula still depended on the superficial similarity of the problems. The similarity between objects in the problems with the same story line was used to instantiate the formulae, so that the objects were assigned to the same variable roles as in the example. Thus, in the same/same condition (e.g. where mechanics chose the cars), performance was higher than for the unrelated group. If the object correspondences were reversed, the same/reversed condition, then performance was lower than in the unrelated condition.

Where the underlying principles were confusable, the superficial similarity of problems with the same underlying structure led to the best performance.

When trying to make an analogy between two problems without an adequate representation of the problem structure, the usual means of instantiating variables through an understanding of what they represent is very difficult. All novices can rely on are superficial similarities. Even when learners are provided with a formula at test, they will still make use of an earlier example in which the principle is incorporated in order to solve the current problem. Ross's results are therefore at odds with those one would expect from a principle-cueing view in which the example plays no role other than as an instantiation of a schema which is either already known or readily induced.

Whereas surface similarity, in the principle-cueing view, is necessary for accessing an example, it is the example's structural features which are used to solve the target. However, Ross (1989a) also found that, in problems with unrelated story lines but corresponding objects, the object correspondences had a large effect on use but none on access. At the same time similar story lines had a large effect on access. This indicates that different forms of similarity differentially affect access and use of a source problem.

The example-analogy view treats a training example as including a kind of recipe which the learner has to follow. It does not assume that the underlying principle can be automatically extracted. When the novice has to solve an exercise problem, the details of the training problem are used extensively to ascertain how a procedure should be employed. For example, the subject solving the SOLO problem keeps to the syntax of the example solution, and simply replaces the values in the source with the corresponding ones from the target. Solving a target problem can therefore be achieved by imitating the sequences of actions that were carried out in the source. Any schema generated from solving problems using an example develops as a *by-product* of the process of generalization. In figure 2.6 the relation between *A* and *B* is partially generalized to *C* and *D*. In creating this partial generalization a partial schema (shaded *S* box in figure 2.6) is created as a by-product. This is represented as a dotted arrow from the generalization line from the source to the target problems.

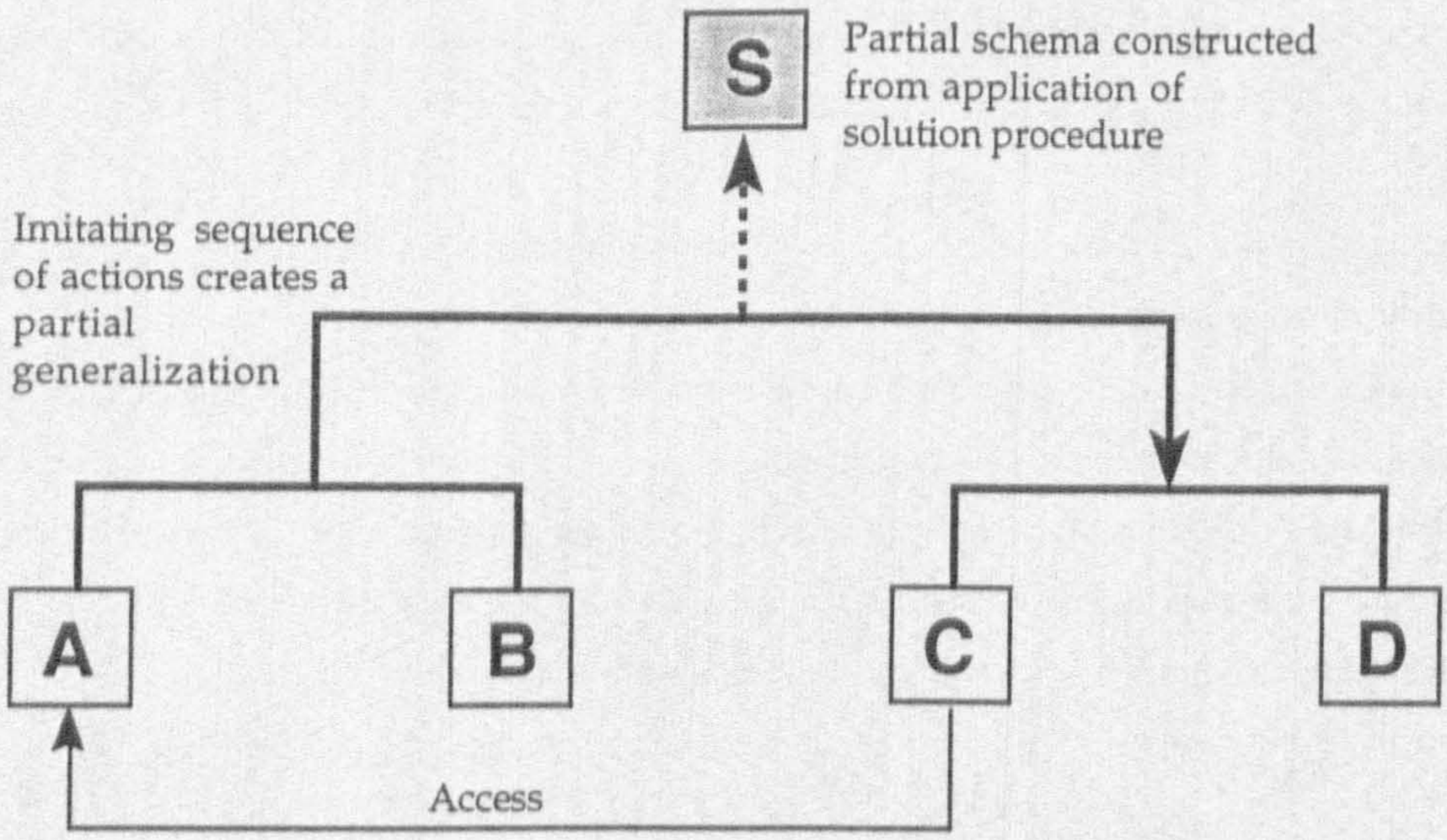


Figure 2.6. The example-analogy view of schema induction.

5.5. Adapting complex problems in unfamiliar domains

If we take analogy as involving the mapping of the structural features of one problem onto another, then a number of interesting points emerge. The main finding of concern here is that people can readily solve versions of a problem that are very similar to each other (close variants), but tend to be unable to solve versions which differ (distant variants). In other words, people have difficulty adapting an example problem in order to solve a new one. One view, which goes back to Thorndike's theory of 'identical elements' (e.g. Thorndike & Woodworth, 1901), is that we cannot expect transfer when there are no similar surface elements, even if two problems share the same underlying features. Since novices can solve close variants of the same problem they must be using some process which does not require an understanding of the underlying structure. They are imitating the example.

A common feature of APS experiments, where successful transfer was found, is that the subjects generally 'understood' the base domain. In the experiments on electricity flow new concepts were introduced by presenting an analogy that was within the experience of the subjects. The underlying relational structures, for example the movement of people through turnstiles and the rate at which they could pass, were relatively simple ones. When between-domain analogies are drawn in expository texts, they are often used as a form of explanation to help the reader re-represent some new concept. Such analogies constitute an intermediate representation. This is not the kind of problem solving that goes on when the novice has only a hazy notion of a new domain, such as science and mathematics. If novices are expected to solve problems based on examples, then they

have to understand them well enough to be able to access and apply the underlying solution structure. Novices, however, rarely understand examples that well.

As was pointed out in section 3.4, *definitions* of understanding emphasize the notion of 'flexibility' - the ability to manipulate or adapt a representation of new information or of an earlier problem in order to solve a new one. This is the kind of understanding that is assumed in APS where the solver is presumed to have a representation of the source problem in LTM, but since solvers find it very hard to adapt solutions to solve novel problems that are different from what was taught, they cannot be said to understand the examples.

In complex problems, transfer is very hard to induce, unless a number of examples are presented, the shared schema is emphasized, and salient structural features are highlighted (Catrambone & Holyoak, 1989; Cooper & Sweller, 1987; Holyoak & Koh, 1987; Larkin, 1989; Novick, 1988; Novick & Holyoak, 1991; Reed, et al., 1990). Transfer can also be induced when the underlying principle is explicitly taught (Perry, 1991) or when the subject is encouraged to 'explain' the relation between the surface and structural features (Brown & Kane, 1988). However, even with some form of hint to use an earlier problem, and with a solution or formula available, students are often unable to adapt the earlier problem to solve the current one (Reed, et al., 1985; Reed & Bolstad, 1991; Ross, 1984; 1987; 1989a).

Reed & Ettinger (1987) attempted to induce transfer by training subjects in the use of tables. There was some transfer when completed tables were given to subjects, who then had only to map the values in the table to the relevant equation. Subjects, however, were still unable to fill in the correct values in the tables despite practice in doing so. In other words, they could not solve distant variants of the problem using the explanations they were given. One of the main reasons was that the explanations still left the subjects with a large number of inferences to make (Robertson & Kahney, 1993).

The relation between example and test problems was examined by Novick & Holyoak (1991). Using algebra word problems they looked at the effects of giving subjects specific numerical mappings for transfer problems. For example, in the 'number mapping hint' condition, subjects were presented with hints such as: 'the 12, 8, and 3 in the band problem are like the 10, 4, and 5 in the garden problem.' Transfer success was much more likely to occur with number mappings than if the subjects were given 'concept mappings' such as: 'your goal in this problem is to arrange the band members into rows or columns so that each row (or each column) has the same number of people in it, with no-one left over. That's like the goal you had in the garden problem of grouping plants into different types so that

there were the same number of plants of each type, with no extra spaces left in the garden.' They found that the numerical mappings were a necessary (but not sufficient) prerequisite for transfer. The difficulty came when subjects had to adapt the procedure to solve a transfer problem.

To appreciate fully the processes going on in problem solving from examples, we have to take account of the problem solver's prior knowledge. Problems in formal domains often require the solver to adapt them in some way. Where problems are literally similar then imitation can lead to a successful solution. However, the more a target problem varies from a source, the more the student has to rely on general domain-specific procedures to adapt the source to fit the target. For this reason Novick and Holyoak found that adapting an example (of an algebra word problem) was correlated with mathematical expertise but not with analogical reasoning ability. Similarly, in a discussion of analogical reasoning in children, Goswami (1992) reviews a large number of studies which fail to take account of the fact that analogical reasoning assumes a degree of prior relational knowledge of the relevant domain on the part of the child.

Furthermore, complex problems involve a large number of interrelated concepts. In the SOLO example, the subject is expected to know about what NOTE does, how CHECK operates, the flow of control represented by EXIT, the differences between the three types of variables '/X/', '*', and '?', what IF PRESENT and IF ABSENT refer to, details of the syntax, and how all of these interact. As Reed et al. (1985) found out in their first experiment, it is asking a lot to expect novices, faced with complex problems for the first time, to remember examples in detail. It is more unreasonable to expect them to adapt them without a great deal of help.

5.6. Summary

Solving problems by analogy assumes that the solver already has a good degree of domain knowledge, can extract the underlying structure of a source problem, and understand it well enough to manipulate it. With regard to novices studying a new subject, these assumptions are unwarranted. Novices do not have a good degree of domain knowledge, or they would not be novices. Principle-cueing assumes that the surface features of problems are the means by which a relevant example is found. Once found, the solution schema implicit in the source can be used to solve the target (depending on how well the novice can adapt it). Novices, however, may not be able to extract the underlying principle from an example. When an example is retrieved, novices may still not understand the principle behind the procedure. Even if they are explicitly presented with the underlying principle or schema, it does not necessarily follow that they understand it or that they can adapt it to a new

problem. As Ross discovered, even when novices appear to understand the principle or formula, they are still strongly influenced by the surface features of the problems. When they are presented with a formula at test, they use the details of the example to solve the target problem because of a lack of understanding of what the variables in the formula represent.

So even if novices can use a procedure accurately, it does not follow that they have a complete understanding of the problem (Perry, 1991; VanLehn, 1989; VanLehn, 1990), or any idea what the underlying schema is. They may have such an understanding but this cannot be ascertained simply from the their performance on a similar problem. With only a weak or partial understanding of the procedure or the underlying principle, successfully solving a problem from an example must involve a different process from the one suggested by the principle-cueing view.

6. Problems facing textbook writers

If we are learning something new, such as a science or a programming language, we may have very little or no prior knowledge of the concepts involved, such as head-tail recursion, entropy, or quantum mechanics. Indeed, Resnick (1989) has argued that some basic scientific concepts are in 'fundamental epistemological conflict with many commonplace everyday conceptions.' For example, technology students find it easy to understand that a book exerts a downward force on the surface of a table but find it hard to understand that the table surface exerts an equal upward force to maintain the book in the same position. The materials scientist Gordon (1976) put it rather pithily when he wrote that, when trying to explain such concepts, we have to take account of the 'pit of anti-knowledge from which materials science has had to extricate itself.' Similarly, natural language understanding can cause confusion in learning some computer programming languages (Bonar & Soloway, 1985). For those reasons, when students have to learn new concepts or are presented with complex examples, past solutions may not be well understood.

Despite our uncertainty concerning the processes that go on when students use training problems to solve exercise problems, textbooks with their training examples still provide the backbone of the teaching and learning that go on in schools, colleges and universities. It is therefore important we get them right. Work by researchers such as Cooper and Sweller (1987), Reed and Bolstad (1991), Reed et al. (1985), Robertson & Kahney (1993), Sweller (1988) has shown that writers have to be very careful about the structure of the problems they provide; and work by Britton, Van, Gulgoz, & Glynn (1989), Britton, Van Dusen, Glynn, & Hemphill (1990), Hiebert & Lefevre (1986), Kieras, 1985, Kintsch (1986), Kintsch & Greeno (1985) has shown that the textual presentation has a strong effect on what will be learned and how well it will be learned .

The writers of textbooks are therefore faced with a dilemma when it comes to how they illustrate the concepts and procedures they wish to convey. Unfortunately the writer can have only a very general idea of the prior knowledge that potential readers will bring with them. Naturally the text will be designed for a specific audience, but even then that audience possesses a wide variety of knowledge, understanding and learning strategies which they will bring to bear on the text laid before them. The subject in the SOLO study had a particular strategy which she employed during her original reading of the text. As she read she consistently tried to find potential problems that might arise and tried to identify gaps in her understanding.

Authors have to make assumptions about how well the readers will grasp new concepts. When they are appealing to the readers' prior general knowledge about the world, they assume that the readers can make the necessary *text-external inferences*. They also have to make assumptions about how much their readers will be able to remember from previous sections. In other words readers will often be called upon to make *text-reinstatement inferences*. Finally, authors have to predict to some extent how well they will be able to apply these new concepts once they have learned them. The SOLO subject had a representation of recursion which was abstract enough to allow her to recognize recursive problems but not specific enough to remember from her original reading of the text how to write a recursive procedure to solve the problem immediately.

The subject successfully solved problems which were isomorphic to examples in the manual. However, she found much more difficulty in solving a problem which required her to *adapt* examples in the book. So even if writers can target successfully a specific readership they are still left with the problem of how to present the material in such a way that it will be easily assimilated, (that is, new material has to be integrated within existing knowledge structures) and readily transferred (the new material can be applied in a different context). In other words the outcome of learning should be 'the ability to use learned information in problem solving tasks that are different from what was explicitly taught' (Mayer, 1989).

So how do writers promote such understanding in their readers? They have only a certain number of pages in which to present a novel concept. If a large number of problems are provided which are similar to each other (the 'near variants' in figure 2.7) then eventually the procedure employed in their solution will become automated. However, that might be at the expense of presenting a range of problems (the distant variants in figure 2.7) which would illustrate the range of applicability of the concept (Cooper & Sweller, 1987). It would be helpful, therefore, to know just what sort of understanding a novice gains when using a training problem to help solve an exercise problem in a novel domain.

Training material can present concrete examples of how a procedure should be used or it can be couched in general or abstract terms. It can present a step-by-step 'recipe' for solving a type of problem or it can try to give an abstract or a hierarchical (goal-subgoal) outline of the solution procedure. The former makes the solution to similar problems fairly straightforward - one simply follows the recipe used in the earlier problem; the latter indicates how a procedure can be applied to a wide variety of problems but at the possible cost of making the procedure difficult to follow.

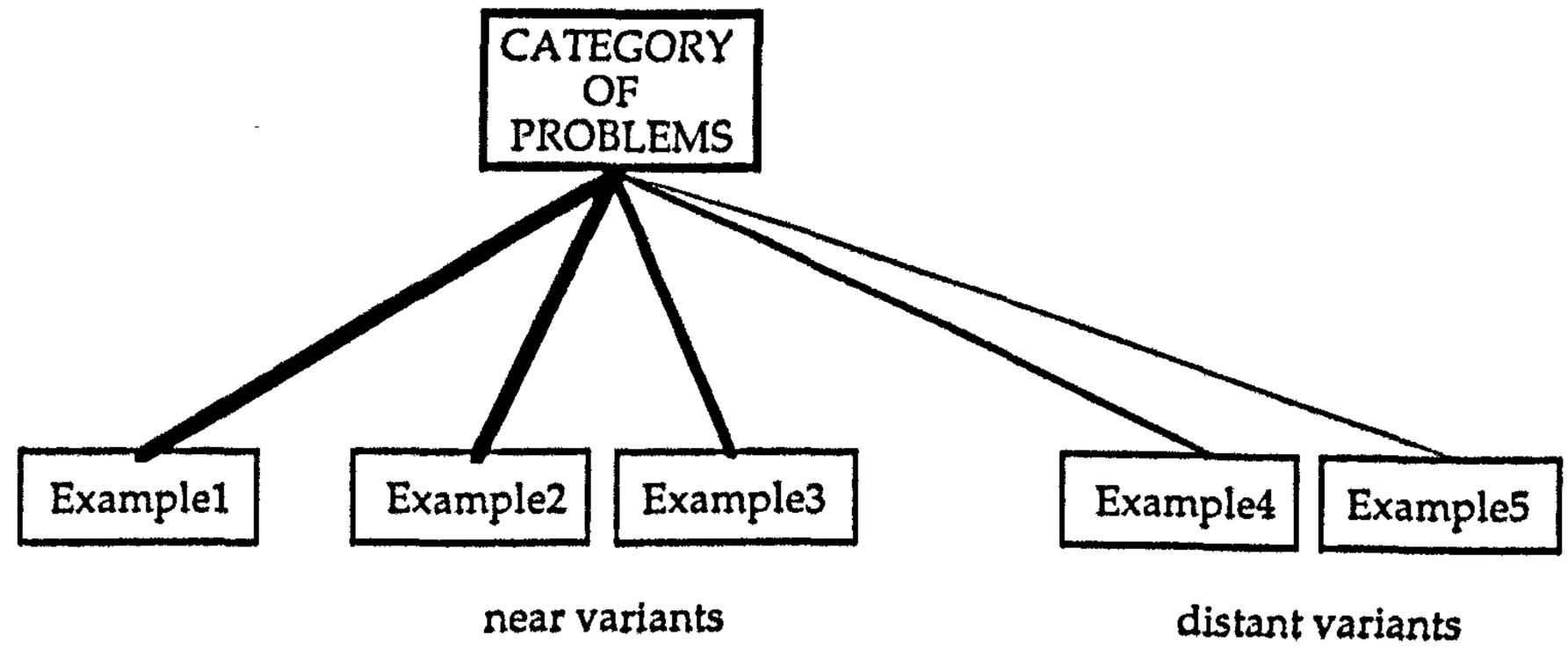


Figure 2.7. Example1 presents a prototype of a category of problems; examples 2 and 3 are closely related and examples 4 and 5 are distant variants of the problem type.

Figure 2.8 represents a category of problems in which the set of problems are related in terms of their solution procedure exemplified in example1. With more distant variants of the problem solvers have to adapt the procedure to a greater extent to find the solution.

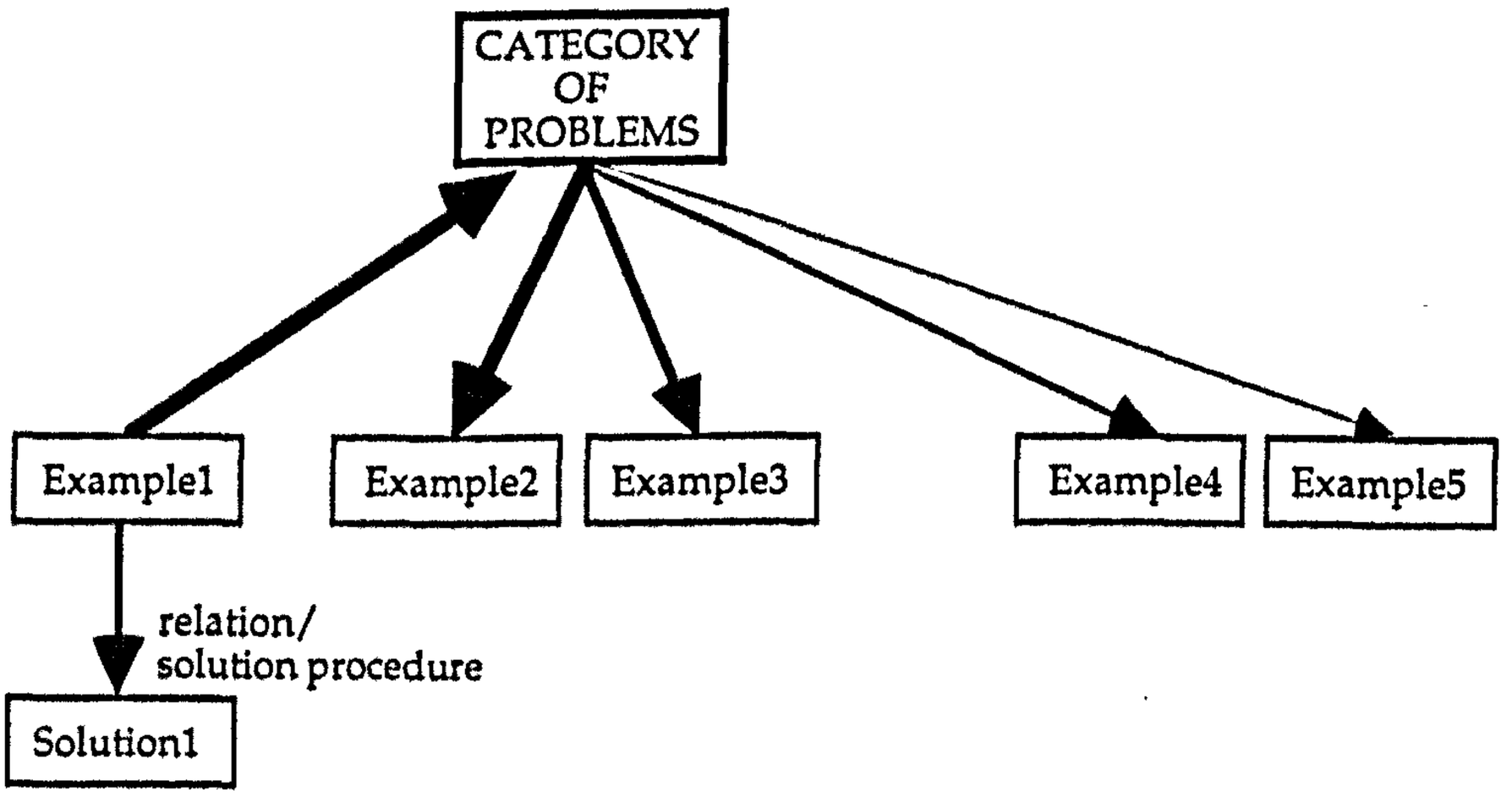


Figure 2.8. A problem category defined in terms of a solution procedure. Example 1 can be seen as a prototype of problems of the same type.

One of the questions writers have to address is just how similar example and exercise problems should be to facilitate learning and what kind of conceptual understanding we should try to instil in the readers.

Furthermore, writers have to provide explicit information about the relationship between training problems and other problems of the same type - usually presented as exercise problems (Conway & Kahney, 1987; Reed, et al., 1974). Since examples provide information about a category of problems, the more information about the features of that category which are given to the reader the better. In figure 2.8 the arrow from example1 to the Category box represents those features of the problem which make it a problem of

that type. If solvers can recognize a new exemplar of that type of problem then they should readily be able to apply the relevant solution procedure. Applying the procedure will tend to be more successful with close variants than with distant ones.

6.1. Do experts have access to their procedural knowledge?

Textbook writers tend to be experts in their field. Because of this, much of their domain-knowledge is 'compiled'. Experts can readily generate inferences that novices cannot make. Skill in textbook writing comes from recognizing those inferences and making the knowledge that generates them accessible to their readership.

In this regard, a rather contentious distinction is often made between declarative and procedural knowledge in terms of their accessibility to conscious introspection. Unlike declarative knowledge, procedural knowledge is often regarded as not being accessible to consciousness (Anderson, 1983; Newell, 1990; Sanderson, 1989). For example, Adelson (1984) states:

> ...knowledge contained in a procedure cannot be inspected directly; what the knowledge is can only be inferred by noting what the procedure does. Having developed these procedures, the information (*sic*) comes to be represented in a way that hides the details of the processing to be done. (p. 495)

However, much procedural knowledge can be accessed and declarative knowledge extracted from it. This is what teachers, coaches, tutors, instructors and textbook writers do all the time. A squash coach, for example, has to break down a complex sequence of movements lasting a fraction of a second into its component parts and typically moves through this sequence in slow motion at the same time verbalizing information about racquet angles and relative positions of various parts of the body. Just how much of this declarative knowledge is inferred from procedural knowledge is a moot point.

If a sequence of actions can be compiled or chunked in this way so that it becomes difficult to access individual pieces of knowledge, then it also makes it harder for textbook writers to access their own knowledge and remove the inferences from it so that it can be fully explained to novices. Textbook writers may therefore make unwarranted assumptions about what a reader knows or can understand without being aware that they are doing so.

In the case of novices, Adelson claims that their knowledge can 'be inferred by noting what the procedure does.' This may not always be so. It is possible for someone to develop a 'syntactic' model for dealing with recursive problems without understanding the

underlying principle of recursion. As has already been argued, novices are capable of imitating and even learning procedures without understanding them. We cannot therefore infer their knowledge from their procedures.

6.2. The role of examples in textbooks

In chapter 1 the subject looked at a number of example problem solutions but looked at the textual explanations only once. This tendency of solvers to look at examples and ignore intervening text is well-documented (Anderson, Farrell, & Sauers, 1984; Anderson, Pirolli, & Farrell, 1988; Chi, Bassok, Lewis, Reimann, & Glaser, 1989; Détienne, 1991; Kintsch, 1986; LeFevre & Dixon, 1986; LeFevre, 1987; Mathews, Buss, Stanley, Blanchard-Fields et al., 1989; Pirolli, 1991; Pirolli & Anderson, 1985; Reed & Bolstad, 1991; Ross, 1987; Ross & Kennedy, 1990; Sweller & Cooper, 1985; Ward & Sweller, 1990; Winston, 1980b). Only if all else fails will some students attempt to re-read the instructions (Gold, 1992). For example, Pirolli (1991) states: 'When a learner is faced with novel goals, the preferred method of problem solving involves the use of example solutions as analogies for the target solution.' VanLehn (1990) goes further: 'examples, exercises and other concrete examples of problem solving are the most salient parts of instruction. The verbal and textual explanations that often accompany such concrete episodes of problem solving have a secondary, indirect effect on learning.'

LeFevre & Dixon (1986) found that students learning a procedural task prefer to use examples as a source of information and that written instructions tend to be ignored. VanLehn (1986; 1990) has built a theory of children's errors on the evidence he has gleaned that people prefer to use examples rather than written explanations. Pirolli (1991) and Pirolli & Anderson (1985) also found that novice programmers relied heavily on examples rather than instructions to help solve LISP recursion problems. Carroll, Smith-Kerker, Ford, & Mazur-Rimetz (1987-1988) redesigned computer training manuals partly to take account of the fact that learners are put off by the 'verbiage' in traditional training manuals. The list could go on.

One reason is that students have expectations about how textbooks are laid out. In formal domains, such as mathematics, science and computer programming, textbooks have a particular stereotypical layout (Beck & McKeown, 1989; Kieras, 1985; Sweller & Cooper, 1985). With experience of such textbooks, students come to develop a schema for that type of text. Such a schema includes the default assumptions that solutions follow statements of the problem rather than vice versa, and that a particular section of a textbook will give them enough information to solve exercise problems at the end of that section. However, it is often the case that textbooks are not structured that way.

6.3. Understanding new concepts: Intermediate representations as an aid to problem understanding

Many concepts in a new domain do not lie within people's normal previous experience. For example, the SOLO subject had only just been introduced to the concept of recursion (at least in relation to AI programs). For this reason it is often useful to provide some *intermediate* means of relating the new concepts to something that the learner does know. People understand new concepts better if they are 'anchored' to existing knowledge schemas. If the text succeeds in providing this anchor then readers will more readily understand and remember new material. One way to explain concepts in textbooks is therefore to try to relate them to the assumed prior knowledge of the reader.

There are several ways open to writers to achieve this. Throughout the SOLO textbook various means are employed to explain the novel constructs that are introduced. Recursion was explained textually in the 1978 edition of the SOLO textbook and was supplemented with diagrams in the 1983 version (Eisenstadt, 1978/1983). Flow of control in SOLO was likened to passing the baton in a relay-race. The subject, however, claims not to have found the analogies used in the SOLO manual particularly useful. Nevertheless, it was the intermediate representations of the 'line-shaped thing' and the 'fan-shaped thing' that allowed her to categorize the problem.

Simon (1984) showed that subjects given analogies understood complex relations between concepts better than those receiving none. The particular form of the analogy has a strong effect on the learners' understanding of the new theoretical domain (Baudet & Denhière, 1991; Brown & Clement, 1989; Gentner & Gentner, 1983). One way of presenting a structural analogy with a novel construct is to give a pictorial representation of the new material. According to Resnick (1989) by providing a different representation of the textual material, writers can 'bootstrap' learners' constructions of novel concepts. 'Objectifying theoretical constructs,' that is, making the abstract more concrete, can be done in texts by presenting the learner with some form of physical display. In that way the theoretical construct can be 'seen'. Thus the fan-shaped and line-shaped representations of iteration and recursion allowed the subject to classify such problems correctly.

6.3.1. Bridging analogies

'Intermediate bridging analogies' (Brown & Clement, 1989), diagrams, graphs and tables are all used to provide an intermediate representation of the material presented in textbooks. Where learners find it difficult to induce the structure of novel abstract constructs, or relate the constructs to the concrete examples in texts (*A* in figure 2.9), they

are likely to find it easier to understand the relation if an intermediate bridging representation is used. In figure 2.9, A represents the relation between a theoretical construct and the concrete example; B represents the relation between the construct and some intermediate representation; and B' represents the relation between the intermediate representation (the representation construct) and the concrete example. These intermediate representations are an important part of the explanation of new theoretical material in textbooks and act as a form of 'scaffolding' to help bridge the gap between the learners' prior knowledge and the new construct.

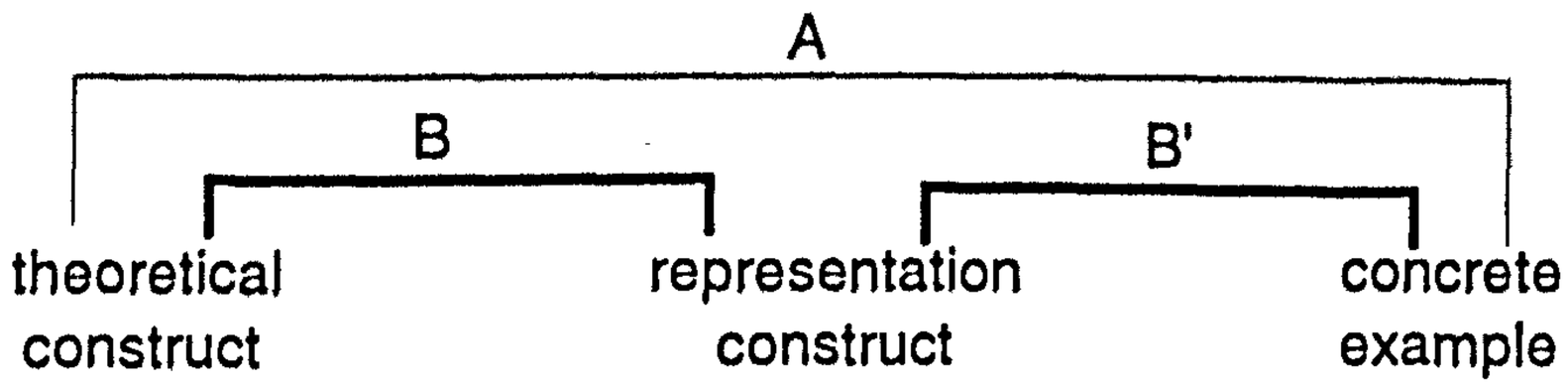


Figure 2.9. Intermediate bridging representations. Where the relation (A) between a theoretical construct and a concrete example is difficult to understand, an intermediate representation may help bridge the gap (B and B').

There have been a number of experiments to test the effects of intermediate representations as an aid in problem solving. Such representations may involve some kind of visual representation (Beveridge & Parkins, 1987; Gick, 1985; Gick & Holyoak, 1983; Lewis, 1989) or an analogy whose purpose is to clarify a concept (Brown & Clement, 1989), or some other way of representing problems such as tables (Reed & Ettinger, 1987). The studies by Gentner and Gentner (1983) and Issing et al. (1989) show that writers have to be very careful about which type of analogy they choose. In some circumstances, such as understanding the flow of electricity through a resistor, electricity is best conceived of as the movement of people. To understand the function of a transistor the flow of electricity is best likened to the flow of water. The effectiveness of the analogy depends on the number of higher-order relations that constrain it.

Both Gick (1985) and Beveridge & Parkins (1987) examined the effects of visual analogues on problem solving and found that visual representations can act as effective retrieval cues. Beveridge & Parkins used both diagrams and coloured strips as cues and the results suggested that the coloured strips of different intensities, representing summative effects analogous to the concepts in the problems presented, was the most effective. Presenting a problem along with a visual representation seems to facilitate recall of a solution. Similarly Gick used a large arrow representing a large force and several smaller arrows arranged in a circle to represent the division of a large force and its convergence on a target. The diagram was presented along with an explanation of the solution to the Fortress problem. When the diagram was reproduced in the target problem, it facilitated

spontaneous transfer. The same was true for subjects who were presented with diagrams alone before solving the transfer problem.

The reasons why pictorial or diagrammatic representations are so effective are discussed by Larkin & Simon (1987). Texts present information in a linear sequence. Understanding this *sentential* representation incurs a great deal of computational cost in terms of search. The larger the data-structure contained within the sentential representation, the greater the search time. In a *diagrammatic* representation the information is indexed by its 2-D location which means that diagrams can make relations *perceptually* explicit which were not so in the sentential representation. Diagrams allow a large number of automatic perceptual inferences, avoid the need to match symbolic labels (matching a variable in one part of a sentential representation to a related variable elsewhere), and obviate the need to search for problem solving inferences.

6.3.2. Providing a schema in texts

The appendix to the SOLO manual builds up to a description of a recursive INFECT procedure. It tries to show how such a procedure can be called to act on a series of nodes linked by the same relation. The text uses a number of analogies to explain how recursion works. It ends by giving a more abstract description of recursion to help the reader understand the general principle behind it. By referring to the 'line-shaped thing' when she identified the problem as recursive, the subject was accessing an abstract representation of the problem type provided by the text. Thus, despite differences in the surface features of the problems she was able to access and use an example in order to solve her problem.

Although providing a general principle, concept or schema at the outset can obviate the need for solvers to use detailed correspondences from the features of an earlier example (Gick and Holyoak, 1980; 1983; Ohlsson and Rees, 1991) it is often the case that such abstract principles cannot be learned directly (Anderson, Farrell and Sauers, 1984; LeFevre and Dixon, 1986; VanLehn, 1990). The usefulness of providing a schema depends on the task facing the learner.

Chen & Daehler (1989) examined the relation between the type of story representation (specific or abstract schema) and positive and negative analogical transfer in children. Where an abstract schema was provided, the subjects were able to transfer analogous solutions spontaneously even when the base and target problems shared few surface similarities. Indeed the abstract representation of the source analogue was a strong determinant of positive transfer. When the target problem involved a solution principle

different from the source, negative transfer resulted. However, although schema training had a strong effect on positive transfer another important aspect was the ability to determine when it should be applied.

The use of intermediate representations can also be justified from a consideration of other aspects of analogical problem solving. By a comparing examples from two disparate domains, one from the domain being studied and one from another well-understood domain, subjects are often able to abstract out the underlying structure or schema from the known domain and apply it to the new one. Some of the benefits of providing an explanatory schema have been listed by Smith & Goodman (1984). They apply equally well to the benefits of diagrams and other pictorial representations such as graphs and tables and can be related to the information processing model of Larkin and Simon (1987). These are:

a) Schemas provide an explanatory framework or 'scaffolding'. They improve understanding since the pre-existing connections between the framework slots can be mapped to the new domain directly. In Larkin and Simon's terms the diagram and text should be 'informationally equivalent' so that information in one representation is also inferable in the other.

b) Schemas contain information that can be added to fill in gaps in knowledge and help form connections between steps. In diagrams this includes the ability to generate perceptual inferences.

c) Schema-based instructions reduce the time required to understand the relation between steps. In diagrams there is less need for search.

d) Schemas boost memory for specific information. According to Larkin and Simon, in diagrams perception permits the reader to focus on perceptual cues and so retrieve problem relevant inference operators from memory.

e) Schemas boost performance where they depend on understanding the relations between steps. Similarly diagrams have computational benefits, since the information in them is better indexed and is supported by perceptual inferences.

f) Schemas should lead to a hierarchical organization of material which should, in turn, lead to 'chunking' and hence to improved recall (Eylon & Reif, 1984). The information in diagrams is perceptually grouped - related bits of information are adjacent to each other.

6.4. Limits to effectiveness of intermediate representations

Writers still have to be very careful that the textual representation of a new construct and the intermediate representation they provide are structurally equivalent. The inferences that readers can make in the known base domain should also be made in the target. If this

is not the case then learners will have difficulty transferring the induced structure to novel problems of the same type.

There is also the question of the function of the intermediate representation. Levin (1989) classifies the functions of 'pictures-in-text' into five categories:

1) *decoration*, where pictures are designed to make a text more attractive but are not related to the content;
2) *representation*, where pictures make the text more concrete, as in children's books;
3) *organization*, where pictures enhance the structure of a text;
4) *interpretation*, where pictures are supposed to make a text more comprehensible;
5) *transformation*, where pictures are presented to make a text more memorable.

Levin then relates these functions to different prose-learning outcomes by appealing to the notion of *transfer-appropriate processing* (Morris, Bransford, & Franks, 1977). Learners have to take account of the goals of the learning context and adapt their learning strategies accordingly. In the context of using pictures in text, Levin argues that writers should use different pictorial representations depending on whether they want to encourage the learner to *understand* the material, *remember* the material, or to *apply* the material. For example, in the studies by Beveridge & Parkins (1987) and Gick (1985) the function of the intermediate representation was to aid retrieval.

An example of an abstract pictorial representation can be found on page 35 of Winston & Horn (1981) where a flow diagram is presented to show how LISP evaluates a particular piece of code (figure 2.10). Its function is therefore interpretative in Levin's sense. Although its function is to help the reader understand how the code is evaluated, the subject BR in a study by Anderson, et al. (1984) tries at one point to *apply* the structure of the diagram to an exercise problems and is momentarily confused. This is because the function SETQ is used in the diagram to *explain* how the code is evaluated but it is not *used* in the original example itself.

Although intermediate representations can be useful pedagogical devices, textbook writers have to be aware of their limitations

6.5. Individual differences

Authors also have to bear in mind the individual differences exhibited by their readers. These differences come about through differences in study processes and strategies, as well as through variations in prior knowledge. In this regard, the role of imitation as a problem solving strategy has been overlooked. Although the notion of imitation is not new

it has remained implicit in many studies of APS, but its importance has been underestimated. Some researchers see it as the resort of 'poor solvers' others see it as a useful strategy that should be taught to all.

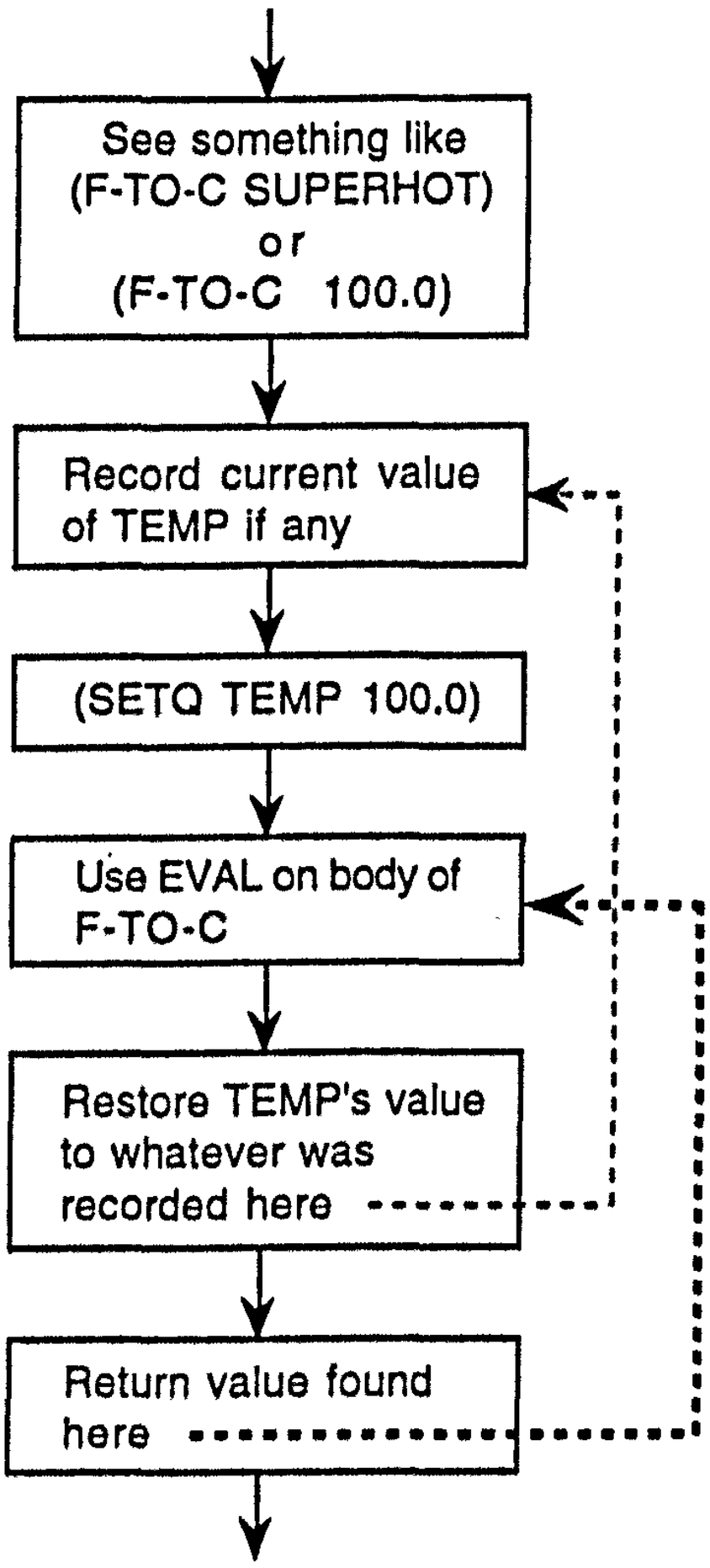


Figure 2.10. Representation of the order of operations carried out when the example on page 34 is evaluated (from Winston and Horn, 1981, page 35

6.5.1. Variations in study processes

Procedures that can help in problem solving can be either specific to a domain or applicable in many domains. The relative merits and demerits of equipping students with different strategies has received much attention (Catrambone, 1990; Cheng & Holyoak, 1985; Détienne, 1991; Green, 1989; Green & Gilhooly, 1990b; LeFevre & Dixon, 1986; LeFevre, 1987). Strategic knowledge is essentially a blend of declarative and procedural knowledge. It is not domain specific knowledge about how to solve a particular problem but provides a general method for finding out how to solve a problem. In other words it

helps organize the problem solving process and provides a general plan of action (Ferguson-Hessler & Jong, 1990; Green & Gilhooly, 1990a; 1990b).

Ferguson-Hessler & Jong (1990) looked at the number of study processes that students used in reading a physics text and the type of knowledge involved in those processes. They found that the range of study processes was the same for both poor and skilled performers but that the difference lay in the predominance of the types of knowledge used. Poor solvers tended to use 'surface processing' by which is meant that they paid more attention to declarative knowledge to solve problems. An example of surface processing was 'reading the text'. That is, the information was processed piece by piece from the shared surface features of problems. This contrasted with the good performers' 'holistic' view of the problems in terms of their deeper structure. An example of deep processing was 'making procedures and assumptions explicit.' This is similar to Chi et al.'s (1989) notion of 'self-explanations'. This type of processing meant that good performers could restructure knowledge to apply it to new variants of a problem because they had good problem schemas, and concentrated more than poor solvers on procedural and situational knowledge. Lacking a good problem model, the 'poor solvers' relied on imitation as a method of finding a solution.

In chapter 1 the subject gave a couple of examples of going beyond surface processing of the text. She attempted to predict what the text was about to tell her. She mentions this explicitly in her study of the text on pages 83 and 84:

> *Subject reads textbook aloud:*
>
> 'Given the restriction to one use of the relation LIKES per node, then only one node could possibly match the wild-card symbol (?) at step 2, and this node became the value of the variable *.'
>
> So obviously we're going to need some sort of recursion to deal with multiple LIKES on page 83.

Here she is 'explaining' the text by generating a hypothesis. She then goes on:

> The main problem with this is that I'm not really being encouraged to think terribly creatively. Because on page 84 it's telling me... em, a new, a new SUSS thing which... involves a bit of recursion. And, if I were concentrating on this properly and having to actually think creatively, I would have thought it out for myself and identified on about page 82, if not about 50 pages earlier, that something like this was going to be necessary.

She expects the text to conform to the strategy she has learned for dealing with expository texts. That is, she wants the text to oblige her to think 'more creatively'.

Chi, et al. believe that 'copying' is the resort of the 'poor performer'. They classified the use their subjects made of the examples provided into 'reading', 'copy and map', and 'compare and check'. They found that 'poor' students used examples to copy elements from it when solving an exercise problem twice as often as the 'good' students. Mapping involved making only a slight alteration to a copied element in the source problem. An example from a geometry problem solving protocol was:
Okay so choosing the axes from the diagram (in the source), *it would be better to tilt it 30 degrees,* (in the target) [coded as mapping to the target].

Imitation was therefore seen by Chi et al. as a strategy used by 'poor' solvers. Other studies, however, have taken a diametrically opposed view.

Green & Gilhooly (1990b) found increased performance in subjects who were *exhorted* to 'copy' a training example, since this appeared to be one successful strategy employed by good students. They examined individual differences among students learning a statistical computing program known as MINITAB. From an analysis of their subjects' protocols, they noted that subjects often selected and copied an example. However, the subjects did not always evaluate the match between the source problem and the target and instead simply copied over the surface features of one onto the other. Both fast and slow learners 'copied' earlier examples when trying to solve a problem but varied in the extent to which they were able to modify the source in order to solve the target. When they exhorted their students to adopt a 'copy procedure' when solving a problem there was an increase in the number of successful solutions.

Green and Gilhooly's exhortation to 'copy' the example does not refer to *how* that example is used. However, their description of the behaviour of their subjects suggests that they were imitating the examples. The subjects were influenced by the perceptually similar features of the source and target problems and inferred that they occupied the same roles in both. Failure to solve the problem in their studies was attributed to a failure to evaluate the match between the corresponding features.

It may well be that the 'poor' students in Chi et al.'s study would not have solved the problem at all if they hadn't used the 'copy' strategy. The results of Green and Gilhooly suggests that the strategy is still a useful one to fall back on.

Novick & Holyoak (1991) make the point that successful problem solving from an example is not necessarily indicative of analogical transfer: 'a student might construct the equation for an algebra word problem but then fall back on general procedures for solving equations that were learned outside the context of word problems. If these procedures were not well-learned, analogical use of the source to construct the equation for the target will not ensure correct solution of the target' (p. 411). Where those procedures are well-learned, they can be used to solve the problem (Bassok & Holyoak, 1989).

There is an interplay between the type of problem solving used and a person's prior domain-relevant knowledge. When problems are isomorphic, imitation will normally lead to a successful solution. However, the more a target problem varies from a source, the more the student has to rely on general domain-specific procedures to adapt the source to fit the target. For this reason Novick and Holyoak found that adapting an example (of an algebra word problem) was correlated with mathematical expertise but not with analogical reasoning.

There are thus limits to the effectiveness of imitation as a strategy. It remains the case though that novices have ipso facto a limited number of domain-specific procedures at their disposal. This makes it likely that imitation will be used by both poor and good students. Imitating an example remains a simple way of solving a problem even for good students. In her original study of the training manual (see Appendix 3a and 3b) the subject fulfilled most criteria for a 'good' student. She could see the underlying similarity between problems with different surface features, but this did not prevent her from imitating the example she found (although she did indicate that she wanted to ensure she knew 'how it worked').

6.5.2. Novices and experts

Consistent with the findings of the novice-expert literature, the subject found it difficult to adapt her knowledge of SOLO to solve a novel problem. At the same time she was not hindered in her problem solving by the surface features of the problems. For example, she showed no difficulty in classifying the problem in terms of its structure. Some aspects of her problem-solving behaviour were at least partly schema-driven.

In trying to turn novices into experts much emphasis has been placed on how to provide the novices with the skills and knowledge that the expert has. That is, models of experts have been used to develop techniques for teaching novices. Studies of expert-novice differences include those of Chase & Simon (1973a; 1973b) who looked at the differences between the knowledge of chess masters, intermediate players, and beginners. They found

that the masters had a greater store of relational patterns of chess pieces, or 'chunks'. The difference between beginners and masters was in how knowledge is organized. In studies of solving physics problems Chi, Feltovich, & Glaser (1981) found that experts understood problems in terms of the underlying physical principles (such as Newton's laws of motion) whereas novices concentrated more on the specific objects mentioned in the problem statement. Furthermore, experts' knowledge is found to be more 'flexible' than that of novices in that it can be readily adapted to novel situations. Flexibility comes about through problem solving experience and the learning of recognizable patterns.

Glaser (1984) provides a summary of the research findings into novice-expert differences when he says: 'Our research suggests that the knowledge of novices is organized around the literal objects explicitly given in a problem statement. Experts' knowledge on the other hand is organized around principles and abstractions that subsume these objects' (p. 98).

Johnson-Laird (1989) says much the same thing when he states that a novice's mental model represents objects and processes that occur in real time; an expert, however, can construct a model that represents highly abstract relations and properties. What this means is that the novice represents problems in terms of their surface features. Step-by-step processes operate on them to transform them into the form of the solution. Novices' mental models are not as elaborated as those of experts since they are unable to generate as many inferences as experts.

Novick (1988) proposed a framework for conceptualizing analogical problem solving to account for the conflicting findings in the literature in which analogical transfer is hard to demonstrate despite the fact that we can readily use a reminded example to solve a current problem. The framework was tested using a number of arithmetic word problems presented to college students. Her framework made predictions about differences in transfer behaviour between experts and novices. She predicted and found that where problems shared the same underlying features but differed in their superficial features then there was more likely to be spontaneous positive transfer in experts but not in novices. When two problems shared the same surface features but their underlying structural features differed, she found stronger spontaneous negative transfer effects among novices than among experts.

The knowledge-based approach exemplified above has emphasized knowledge differences between experts and novices. However, the fact that experts know more than novices does not help us understand how real novices solve problems. Instead, it might be more enlightening to examine differences in processing knowledge. Larkin, McDermott,

Simon, & Simon, (1980) looked at the problem solving strategies of experts and novices and argued that experts 'work forward' whereas novices 'work backward'. That is, experts generate hypotheses using the information in the problem; their problem solving behaviour is schema-driven. Novices work backward from the goal and their behaviour is search-driven. But if the same problems are given to novices and experts to compare their solution processes then it is hardly surprising that experts perform far better on a variety of measures such as time to solution or chunk sizes. When problems become hard enough for only experts to solve (the kind of problems they normally have to deal with) experts, too, use a search-driven pattern of behaviour.

6.5.3. Understanding and intelligence

It may be that poor students fail to solve problems and to make necessary inferences due to a lack of what the psychometricists have called 'intelligence'. Nevertheless, it would be unfair to suppose that someone attempting to solve a problem without all the relevant knowledge is not intelligent. Only when the solver possesses all the relevant knowledge to solve a particular problem can one ascertain that the solver is acting with intelligence or not. If the solver can form a problem model then that person can be said to 'understand' the problem. In this sense both Kintsch (1986) and Ohlsson and Rees (1991) would agree when the latter state that 'a problem solver acts with understanding when a problem solution is developed in the context of relevant knowledge.'

Textbooks attempt to provide a 'context of relevant knowledge'. If understanding arises within that context then a solver is not only acting with understanding but also with intelligence since the latter can be defined as the extent to which one uses all one's knowledge to achieve one's goals. Understanding, in the sense given by Ohlsson and Rees, and intelligence as it is described by Newell (1990) are virtually synonymous. Here, for example, is Newell's definition of an intelligent system:

1. If a system uses *all* of the knowledge it has, it must be perfectly intelligent. There is nothing that anything called intelligence can do to produce more effective performance. If all the knowledge that a system has is brought to bear in the service of its goals, the behaviour must correspond to what perfect intelligence produces.

2. If a system does not have some knowledge, failure to use it cannot be a failure of intelligence. Intelligence can work only with the knowledge the system has.

3. If a system has some knowledge and fails to use it, then there is certainly a failure of some internal ability. Something within the system did not permit it to

> make use of the knowledge in the service of its own goals, that is, in its own interests. This failure can be identified with a lack of intelligence.
> (Newell, 1990, p. 90).

In a novel domain it is therefore important to provide all the knowledge necessary to solve problems in that domain. This, of course, is not to guarantee that they will be solved, merely that it makes it more likely. For this reason it is unreasonable to oblige novices to make too many inferences in a novel domain since making inferences assumes some prior knowledge which the novice may not have.

6.6. Summary

Textbook writers have to make assumptions about the readers' ability to make both text-reinstatement inferences and text-external inferences. The former depends on how much the reader can remember from earlier parts of the text. The latter involves assumptions about the reader's prior knowledge. In most cases the readers' ability to make these inferences is an empirical question. Nevertheless, authors can make it easier for the reader to understand new concepts and procedures by reducing as far as possible the number of inferences the readers have to make. This, in turn, depends on the writer's ability to access his or her declarative knowledge.

Writers also have to be aware of the readers pre-existing schemas for how textbooks are constructed. This means that they have to be cognisant of the fact that readers tend to ignore the intervening text and concentrate on the example solutions when trying to solve exercise problems. The structure of the example solutions and the explanations surrounding them are therefore of great importance.

Many methods are open to authors for explaining new concepts. These include diagrams, graphs, analogies, and so on, that allow the readers to re-represent the concepts in terms that they understand. Most forms of intermediate representation make it easier for novices to assimilate new concepts. They help the reader to access the shared underlying structure of problems and new concepts, and can act as a guiding principle to aid problem solving. When an explanatory schema is presented along with an example, novices are more likely to be able to adapt the example to solve distant variants of a problem type.

Finally there are bound to be individual differences in the readers' ability to understand new material. Even so, hints to use particular examples when solving exercise problems should help both poor and good solvers by removing the need to search through training material for a relevant source to imitate.

7. Learning from examples

Having successfully solved the FLOOD problem using the INFECT solution, and having attempted a second problem involving iteration, the subject in the SOLO study was able to solve a third problem involving both without any difficulty (see Appendix A.3.3.4). She was also able to generate another isomorphic recursion program, the BULLETHOLE problem (Kahney, 1982), easily and without further reference to the training manual. She had, it seems, induced the structure of simple recursion problems in SOLO.

In problem-solving, as in other areas involving concept or category representation, people generate inferences about how a concept, such as recursion, is instantiated in particular examples. Learners can then induce inference rules or theories which make clear in what conditions or situations a particular action must be taken. These inference rules are at first situation-specific but after solving several examples a learner will come to generalize across examples (Anderson, 1983; Chi, Bassok, Lewis, Reimann & Glaser, 1989).

At first sight it may seem unclear how one can learn anything from a process of imitation. That is, if we are unaware of the structure of a problem type, how can we learn about it by simply copying the procedure which an example solution embodies? To some extent this point has already been addressed in the discussion of the learning model proposed by Newell and Simon (1972). Understanding of the logical justification for a procedure arises independently of the mechanization of the procedure. However, we also have to explain how we derive a solution schema at a level of abstraction which can eventually be applied to distant variants of a problem type. The principle-cueing view of schema abstraction assumes that we can readily induce a problem schema, such that the examples from which it was derived play no further role in subsequent problem solving. However, other views see abstraction as much more piecemeal, with schemas retaining much context-specific information. Context specificity is predicted from an IPS viewpoint, since it regards problem solving as being heavily influenced by the surface features of problems. It therefore takes time to learn solution schemas at different levels of abstraction.

7.1. Generalization and learning

'(Deduction) is impossible unless a man knows the primary immediate premises ... We must get to know the primary premises by induction; for the method by which even sense-perception implants the universal is inductive...' (Aristotle, Posterior Analytics, Book II, chapter 19, c. 330 BC.).

From an empiricist viewpoint our knowledge of the world is constructed from our experience of individual episodes. Similarly, our concepts are usually derived from our experience of individual cases. In order to derive general knowledge and beliefs which are sufficient to make reliable predictions about events, we have to make use of inductive inference. Given a number of specific problem solving episodes we somehow construct general procedures for dealing with this category of problems in the future. The general knowledge derived from our inductions then form the premises of our deductions about the world.

7.1.1. Induction and transfer in the principle-cueing view

According to the principle-cueing view, when solvers access a source problem which exemplifies a principle or concept, they go on to use that principle or concept to solve the current problem with no more need for the earlier problem. When a relation or set of relations that holds in one domain can be extrapolated and applied to another, then this generalization process abstracts out the common underlying properties of the two analogues. These abstracted properties constitute a schema for further problems of this type. Gick and Holyoak (1983) provide a table illustrating the correspondences between the Fortress and Radiation problem with its underlying schema, reproduced in table 2.3.

Fortress problem	
Initial state	
Goal	Use army to capture fortress
Resources	Sufficiently large army
Operators	Divide army, move army, attack with army
Solution plan	Send small groups along multiple roads simultaneously
Outcome	Fortress captured by army
Radiation problem	
Initial state	
Goal	Use rays to destroy tumour
Resources	Sufficiently powerful rays
Operators	Reduce ray intensity, move ray source, administer rays
Constraints	Unable to administer high-intensity rays from one direction safely
Solution plan	Administer low-intensity rays from multiple directions simultaneously
Outcome	Tumour destroyed by rays

Convergence schema	
Initial state	
Goal	Use force to overcome a central target
Resources	Sufficiently great force
Operators	Reduce force intensity, move source of force, apply force
Constraints	Unable to apply force along one path safely
Solution plan	Apply weak force along multiple paths simultaneously
Outcome	Central target overcome by force

Table 2.3. Correspondences between two convergence problems and their schema. From Gick and Holyoak, (1983)

In figure 2.11, A1 and A2 are two problem statements (the Fortress and Radiation problems, say), B1 and B2 are their respective solutions. The line linking the A terms and B terms represents the relation (the solution structure) between them. The solution structure for the Fortress problem corresponds to the solution plan in Table 2.3, for example. The act of forming a generalization between two problems (represented by the line linking A1 and B1, and A2 and B2) leads to the creation of a schema, in this case the 'convergence' schema (represented by the white *S* in figure 2.11). According to Gick and Holyoak (1983) two analogues were enough to induce a schema, although Catrambone and Holyoak (1989) found that presenting three similar problems was more effective in inducing one.

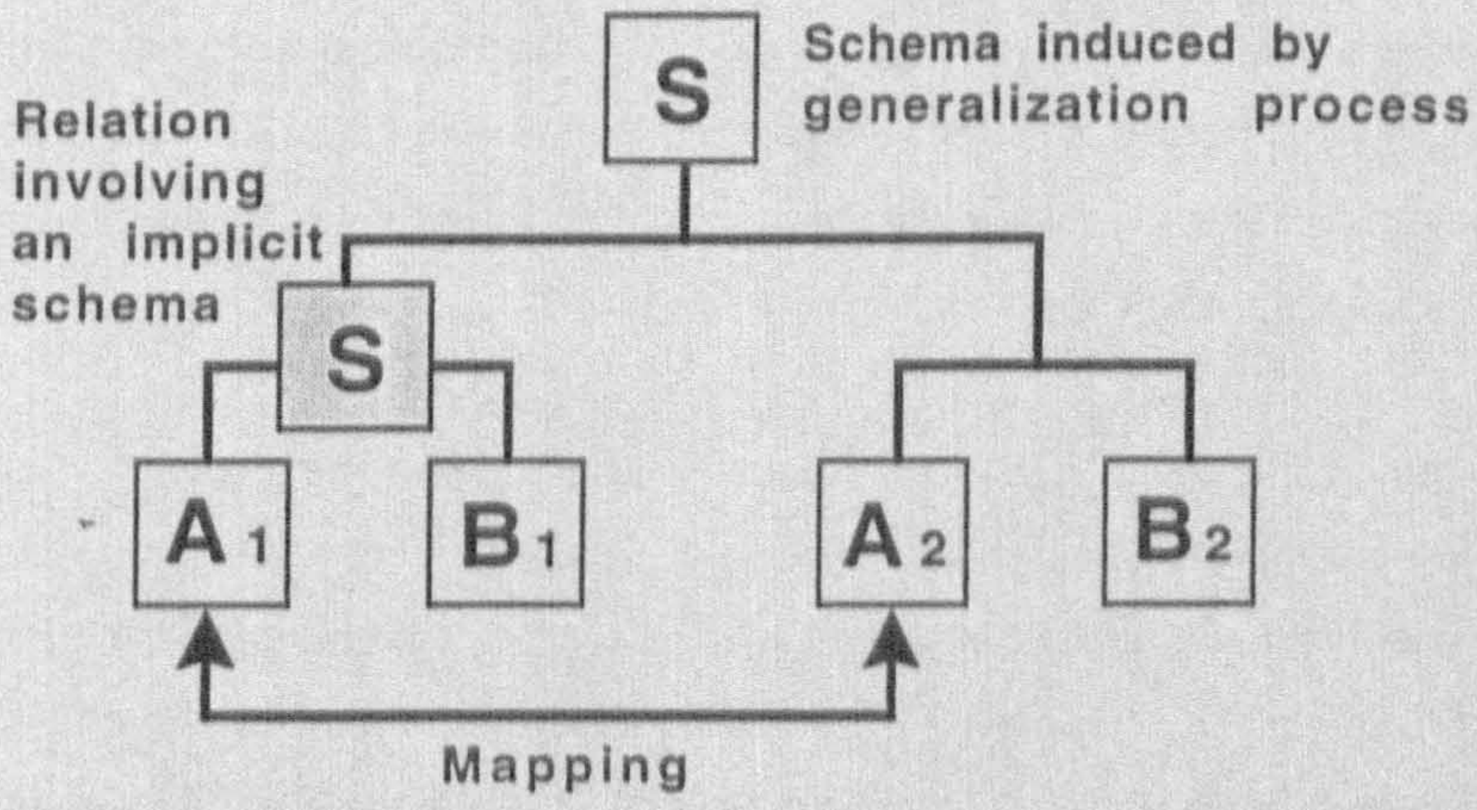


Figure 2.11. Schema induction from an example and a test problem.

Catrambone & Holyoak presented subjects with several analogues to a superficially dissimilar target problem. Subjects were asked to compare the analogues prior to solving the target problem. Even without a hint, a significant degree of transfer was obtained. By comparing 'convergence' problems (represented in figure 2.11 and 2.12 by A_1 and A_2), subjects were presumed to be able to abstract out the implicit schema in the problem (the

shaded S box) so that the schema now becomes 'explicit' and can be applied directly (the white S box in both figures).

The induced schema is then assumed to be used to solve further problems of this type. So, when a further test problem is presented (the C box in figure 2.12), subjects can thereafter access the solution schema directly without any need to access the previous examples.

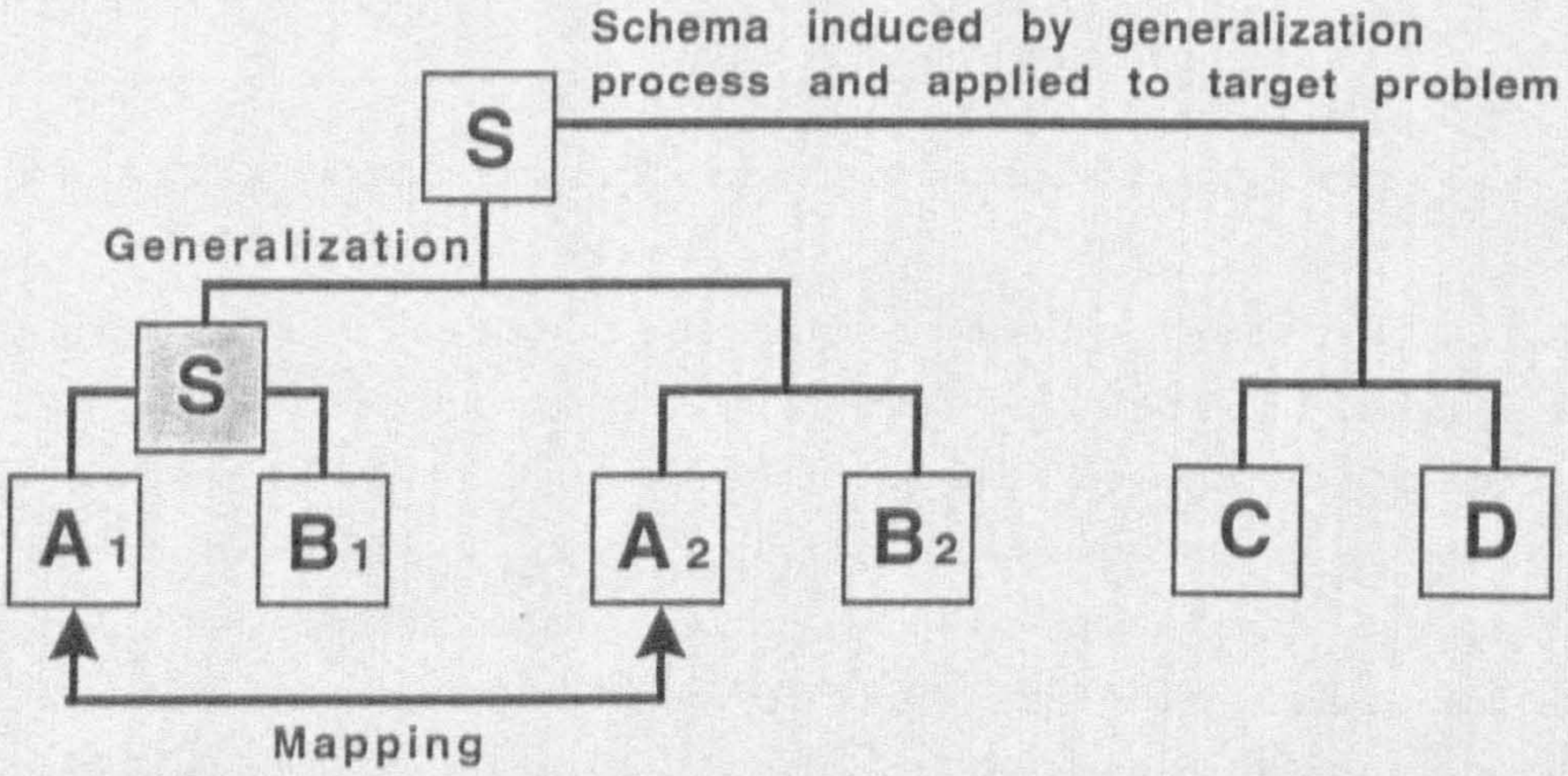


Figure 2.12. Schema induced from two example analogues and applied to a target.

However, these effects all but disappeared when the examples were presented in different contexts. Only when the structural features of the problem were highlighted in the analogues (by rewording the questions and asking the subjects questions which focused on those features) did prehint transfer reappear. Prehint transfer was further enhanced by providing three analogues as opposed to two.

For Holyoak learning can be said to have occurred when a schema has been induced which is capable of representing a particular problem type. When presented with a novel problem, the learner can categorize it as an instance of a class of problems because of the schema induced from earlier examples. (Experts are thus better able to retrieve and use analogies from other domains since they are better able to focus on causally relevant features of the target to use as retrieval cues.)

This form of generalization assumes that schema induction is a straightforward process and that the subjects have all the information they need to build the relevant schema. If learners are unaware of the structural aspects of a problem, or if they are not given a hint, then each problem solving episode will be isolated from the others. Furthermore, in Gick and Holyoak's examples the schema is relatively 'transparent': there are only a few salient objects (e.g., army, fortress; rays, tumour) and effectively two main operators

('divide' and 'converge'). Although abstraction of structural information *can* occur from exposure to only one example, most of the time it does not. Many problems facing novices are much more complex than the 'convergence' problems and it takes some time to learn the solution schema (Cooper & Sweller, 1987).

7.1.2. Schema acquisition in the example-analogy view

An alternative acount of schema acquisition is given in the example-analogy view in which a schema is created as a *by-product* of generalization. In this view, the generalizations that novices are likely to make are limited and contain much context-specific as well as problem-specific information. This is not to say that people are not capable of abstracting out deep structural principles from exposure to only one example of a problem type. Clearly some are. However, the abstract principles and the surface features can co-exist. According to the principle of *conservative induction* (Medin & Ross, 1989), we retain much specific information about exemplars. However, that does not prevent us from generating theories about the range of applicability of a concept (Barsalou, 1989; Brown, 1989; Collins & Gentner, 1990; Medin & Ross, 1989; Rips, 1989).

Learning, according to Ross (1984; 1989b), comes about when the information gained in the application of an analogy is generalized over. It is in the actual process of applying an analogy that a generalization occurs. Partial generalizations may form the beginnings of problem schemas. With repeated exposure to analogous problems the information gained becomes partially decontextualized, and generalized goal, action and feature information is extracted. Thus, knowledge about a problem type is *incrementally refined*. Learners have therefore at their command schemas at various levels of abstraction from high levels, which may at times be difficult to access, to low levels, which make access easier and may provide information about how a principle at a higher levels may be used. On this view, the subject described in Chapter 1 was able to access and use the INFECT example to reconstrue her representation of the FLOOD problem and generate a solution.

From another point of view, in a series of experiments on categorizing, Whittlesea (1987) concluded that the abstraction of general information from a series of instances is a by-product of the demands of particular tasks and that performance is under the control of memory for specific experiences rather than of general knowledge. Abstraction is therefore a passive process which arises by *not* attending to particular properties in the tasks. The same process occurs in studies of implicit learning. According to Mathews, et al. (1989), passive abstraction means that conscious thinking is not necessary to extract the regularities needed for performance on complex tasks.

The picture that emerges from these views of generalization is represented in figure 2.13. In this model the schema is seen as arising as a by-product of the generalization of the problem solution from one problem to another. Aspects of earlier problems (*A1, A2, A3* in figure 2.13) are accessed. The relation between the problem statements (the *A* boxes) and the solution (the *B* boxes) is abstracted out and generalized over and applied to another problem of the same type. For example, the line linking *A1* and *B1* represents the relation between the problem statement and solution. This underlying relation is abstracted out and applied to the *A2* problem to generate the solution, *B2*. However, this generalization is only partial, hence the dotted line linking *A1* and *B1* to *A2* and *B2*. Any schema induced from this contains much episodic information, hence the dark *S* box in the figure. Later problems may remind the solver of aspects of specific examples as well as of some abstracted out elements in the solution schema, hence the access arrows to both the *A* and *S* boxes.

The schema abstraction process is therefore different from that described in figure 2.12 where the schema is formed as part of the generalization process. The schemas (*S* boxes in figure 2.13) become increasingly transparent representing the fact that knowledge becomes incrementally refined with exposure to problems of this type. The solver still has access both to specific problems and to schemas at different levels of abstraction.

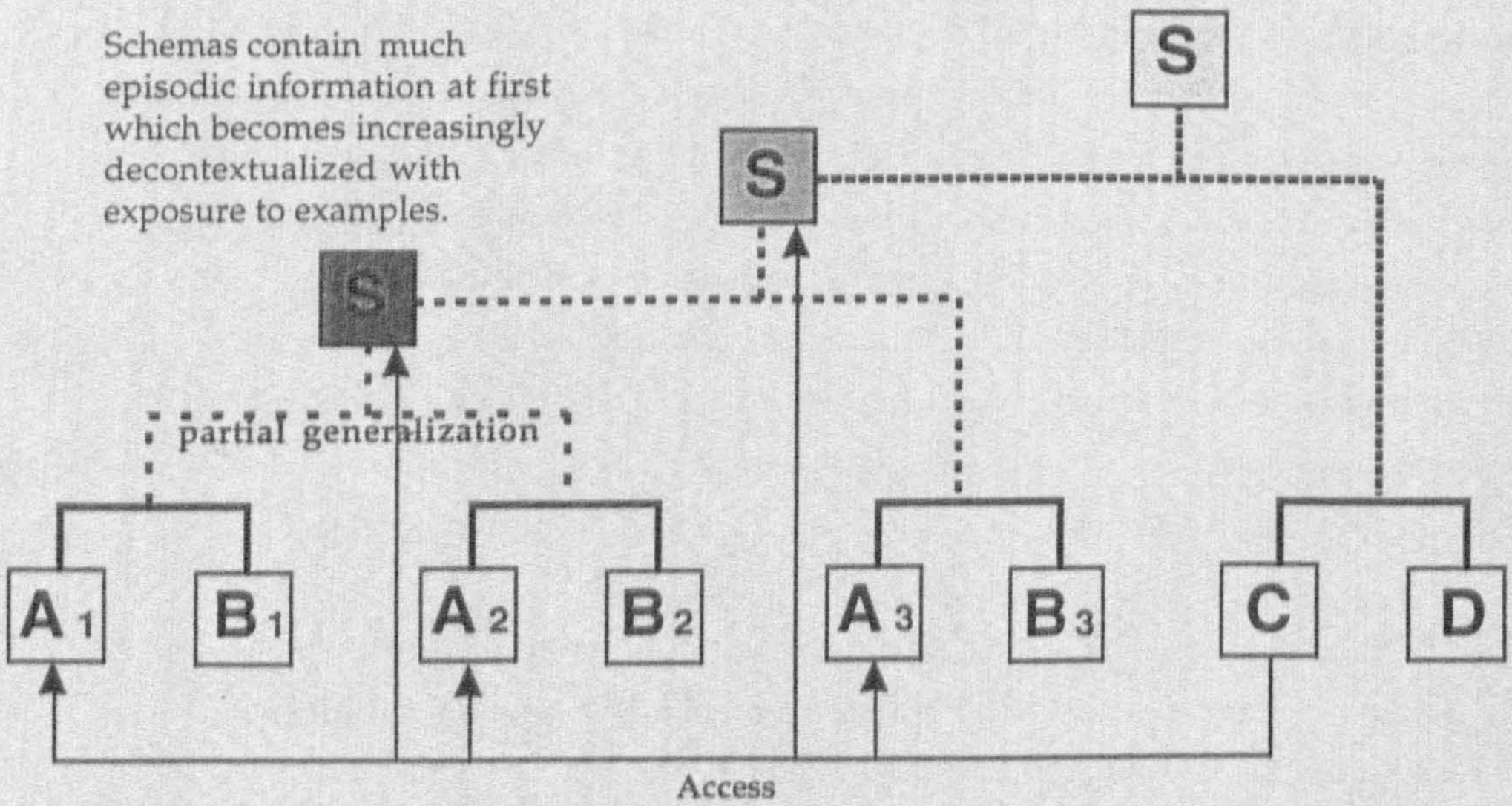


Figure 2.13. Generalization and schema formation according to the example-analogy view.

7.1.3. Abstraction and conservative induction

Medin & Ross (1989) make 3 claims with regard to concept formation which are equally applicable to learning problem categories. The first refers to case-based reasoning. They

argue that reasoning is often based on specific instances. If we classify a novel animal as a mammal then it is not by reference to some abstract concept such as 'mammalhood'. Instead the features of the novel animal may remind us of some other mammal which is already known to us. A lynx, for example, may remind us of a cat.

In contrast, learning models such as ACT*, SOAR and those of Carbonell (1983) and Holyoak (1985) assume an automatic abstraction process which is independent of the specific examples on which it operates. Medin and Ross, however, claim that abstraction is not necessarily an autonomous process, but appears instead to derive naturally from exemplar comparison and use. This is a view shared by Dellarosa-Cummins (1992).

Medin & Ross's third claim is that induction preserves a great deal of information about the specific examples from which the induction is derived. Spencer & Weisberg (1989) and Catrambone & Holyoak (1989) found evidence for redundant and irrelevant specific information in whatever schema subjects had derived from the examples presented. This is what Medin and Ross mean when they say that induction is conservative.

> ...we argue that specificity may make access to and application of relevant knowledge easier, may permit more graceful updating of knowledge, may protect the cognitive system from incorrect and inappropriate inferences, and may provide just the sort of context sensitivity that much of our knowledge should, in fact, have. (pp. 190-191).

If this is the case then we would expect context to exert a strong influence over the ability to generalize and transfer solutions from one problem to another. It also means that solvers will keep very closely to the 'recipe' included in an example problem. Conservative induction explains why imitation is a useful and often necessary problem solving strategy, which nevertheless leads to the induction of solution schemas.

7.2. Implicit and explicit learning

When a subject is presented with a number of exemplars of a problem type, the implicit schema relating the problems eventually comes to be learned. Evidence for a type of unconscious learning has come from studies of implicit learning. 'Implicit knowledge results from the induction of an abstract representation of the structure that the stimulus environment displays, and this knowledge is acquired in the absence of conscious reflective strategies to learn' (Reber, 1989, p. 219). This type of learning can arise even though the relations between features of the task environment are quite complex.

One early study of implicit learning processes came from Posner & Keele (1968). They presented subjects with a series of dot patterns which were graded distortions of a number of 'prototypical' patterns of dots. These dot patterns ranged in how far they varied from their respective prototypes. Where they varied widely, the subjects took longer to learn the prototype but produced greater transfer than those who learned the prototype from patterns which varied little from the prototype. In a follow-up experiment (Posner & Keele, 1970) it was shown that the concept or schema learned by the subjects was recalled far better than already presented specific instances of distortions from the prototype. From the examples provided, subjects induced an abstract representation of the prototype which could be used to classify new instances. A summed features view of categories would have created a blob rather than a useful prototype (Reber, 1989).

Other studies have also found that concept formation appears to result as a by-product of experience with individual instances. Fried & Holyoak (1984), for instance, found that categories could be learned without the subjects' being told the category label of each instance. Even in the absence of instructions to learn the category and without feedback about errors, they still succeeded in learning the category. From their study they developed a category-density model of classification from individual instances which incorporates three assumptions:

1 That highly salient features of presented instances are encoded initially. These instances are regarded as a sample from which we can estimate how widespread are the properties of potential exemplars. Only later are subjects able to search for less salient features that are more diagnostic of category membership.

2 With experience of category exemplars and the distribution of exemplar properties, new exemplars can be classed along the lines of Bayesian probability (see also Anderson, 1990).

3 People assume that categories have a central tendency and a certain degree of variability. Presented instances are then used to revise any original assumptions about such variability around a mean. This in turn affects how the next presented item is evaluated.

The ideal outcome of category learning is a representation of the 'dimensions of variations among category exemplars' (Richardson & McCarthy, 1990). The ideal outcome of teaching concepts using examples in textbooks is to point out the range of applicability of a concept or procedure - the variablilty of a concept or procedure from a prototype or central tendency. Only then can one have a mental representation which is 'flexible'. Learning about the adaptability and applicability of a concept or procedure may come about chiefly through experience with individual instances. However, we can learn how to classify instances by simply being told a concept (Ohlsson & Rees, 1991). Trying to apply a

concept in a domain in which one is a novice is effortful. When students are *told* a concept, they often have to be continually reminded to use it in specific cases. When a concept is formed from the *experience gained by using it*, students are more likely to know when it can be applied to new instances.

7.3. Summary

Problem category induction is an automatic process that arises from exemplar comparison and use. Understanding of the logical justification of a procedure comes about automatically from the process of generalization. Induction is conservative in that much specific information is retained from earlier examples. With continued experience of a class of problems, learners derive a schema which becomes increasingly decontextualized and incrementally refined. In this way, novices learn the range of applicability of new concepts and procedures, and no longer have to rely on specific examples. Nevertheless, the principle of conservative induction ensures that we retain information about specific examples which can sometimes be used when needed. Indeed, Détienne (1991) has shown that experts use both schemas and specific prototypical examples in solving new problems.

8. AI models of analogical problem solving

Since analogy is regarded as a powerful and ubiquitous learning and problem solving mechanism, several AI models of analogy have been developed such as PUPS (Anderson, Boyle, Corbett, & Lewis, 1990), ACME (Holyoak & Thagard, 1989), PI (Holland, et al., 1986), and SME (Falkenhainer, et al., 1989). AI models of APS are often very successful in that they solve problems by analogy to examples in much the same way that humans are capable of. That is, they are often good models of human analogical *competence*. The aim of this section is to explain why they are not necessarily good models of human analogical *performance*. They may model what humans are capable of, but they don't necessarily do what humans do.

Some AI models of APS have been criticized for representing the world in such a way that the model will operate successfully upon that form of representation. This criticism was levelled at the PI program by Dejong (1989) and Palmer (1989) (see section 5.2.1). Gentner's Structure Mapping Engine has been similarly criticized. It relies on a hierarchical propositional representation of the system of relations between problem elements. However, the particular predicate structure chosen to represent a specific situation 'represents choices made by the theorist, who makes the important decisions about syntactic parsing' (Goswami, 1992). Palmer (1989) gives examples of a higher-order relation such as GREATER-THAN [TEMPERATURE1 (object1), TEMPERATURE2 (object2)] and CAUSE [PUSH (object1, object2), COLLIDE (object1, object2)]. He argues that they could equally well be represented as lower-order ones, such as: HOTTER-THAN (object1, object2) and CAUSE-TO-COLLIDE-BY-PUSHING (object1, object2).

8.1. Mapping in PUPS

According to Anderson, simulating analogies in ACT* had proved 'awkward' (Anderson, Farrell & Sauers, 1984; Anderson, Pirolli & Farrell, 1988; Pirolli & Anderson, 1985). For this reason, in his more recent computational model of human knowledge representation known as the PenUltimate Production System (PUPS: Anderson, 1989; Anderson, Boyle, Corbett, & Lewis, 1990; Anderson & Thompson, 1989), knowledge is represented as schema-like structures which contain three obligatory 'slots':
an *ISA* slot specifying the category of which the structure is an instance;
a FUNCTION slot specifying what function the structure fulfils;
a *FORM* slot specifying the form of a structure.
Such representations also include optional slots specifying other prerequisites, such as the

particular context in which the representation takes place. The representation of a LISP function would take the following form (from Anderson and Thompson, 1989):

structure 1
ISA: function-call
FUNCTION: (add 2 3)
FORM: (list + 2 3)
CONTEXT: LISP
MEDIUM: CRT-screen
PRECONDITION: CONTEXT: LISP

This is a declarative statement which states that the structure *isa* function call whose *function* is to add 2 and 3. Its *form* is a list containing a '+' followed by a '2' and a '3'. Together these form the three obligatory slots. There are also three further optional slots asserting that the *context* is LISP and that it is being entered through the *medium* of a cathode-ray tube screen. The final *precondition* means that the context (LISP) is necessary for the form to achieve the function.

The form and the prerequisites together *imply* the function. This is a form of inductive inference or 'implicational semantics' and means that the above example can be represented by the implication:

IF the goal is to achieve the function (add 2 3)
and the context is LISP
THEN use the form (list + 2 3)

PUPS is used primarily to simulate within-domain analogies and does not deal with metaphor or the types of analogy studied by Gentner, for instance. PUPS structures encode the relations between the function (what it does) and the form (how it does it) of a problem; that is, it specifies the solution structure.

Learning in PUPS involves inspecting the trace of an analogy, and building productions directly reproducing the effects of that analogy. Thereby the implications originally induced to perform the analogy are stored. This occurs with first the successful analogy leading to marked improvement in speed and accuracy in further examples. 'Our view is that this knowledge compilation occurs simultaneously with the first successful analogy. Subsequent occasions where the knowledge is required show the benefit of the compiled production' (Anderson & Thompson, 1989, p. 284). Knowledge compilation builds new productions which directly produce the effect of the analogy without further reference to

the example; that is, it encodes how items are structurally related. It therefore builds a schema for a problem type.

Anderson & Thompson (1989) illustrate analogical problem solving in LISP using the PUPS architecture. Let us suppose that we have two LISP structures the first of which is a target structure (structure*x*) representing a problem requiring solution, and the second of which is a source example (structure*y*) retrieved from LTM. The context is LISP and the function (the goal) is to extract the first element of a list:

structure*x*

isa: function-call

function : (extract-first(a b c))

form: ???

structure*y*

isa: function-call

function: (extract-first (p q r s))

form (list car '(p q r s))

According to Anderson & Thompson, the required analogy is therefore of the form:

function(structure*y*) : form(structure*y*) : : function(structure*x*) : ???

or more specifically:

(extract-first(p q r s)) : (list car '(p q r s)) : : (extract-first(a b c)) : ???

The required solution involves putting **(list car '(a b c))** in the form slot of **structure*x***. To do this the following mapping is created:

list	-->	list
car	-->	car
'	-->	'
(p q r s)	-->	(a b c)

For a mapping to take place the first elements in the two function slots must correspond. If that is successful all other elements (the indeterminate correspondences) in the function slots are put into correspondence **((p q r s)** and **(a b c))**. It is a necessary and sufficient

condition for an analogy to be successful that mappings be found for all terms in the function slot.

However, it is not clear that this example as presented by Anderson & Thompson constitutes an analogy. In Gentner's terms the two problems are 'literally similar.' They both have shared attributes and relations. Only the value of one attribute is altered from **p q r s** to **a b c**. Anderson & Thompson have therefore described imitative problem solving. Although their brief description of this particular example of APS in PUPS constitutes a description of imitation, what PUPS actually does is quite different. PUPS solves problems by mapping a structure from a source to a target. To see how it performs its analogical mapping we will look at a similar example involving **car**.

Anderson, Boyle, Corbett, & Lewis (1990) provide an example of solving a similar problem by analogy in PUPS. An example is given ('**example1**' below) to demonstrate how to extract the first element of the list '**(fast computers are nice)**. The solution is to use **car** in the structure (**car '(fast computers are nice)**) which would give **fast** as its answer. To 'understand' the example PUPS has information about the relations between elements of the solution structure:

car
ISA: function
FUNCTION: (function-in **car-structure**)
FORM: (text **car**)

car-structure
ISA: lisp-code
FUNCTION: (calculate-first **arg**)
FORM: (list **car arg**)

example1
ISA: lisp-code
FUNCTION: (illustrate **car**)
(calculate-first **lis**)
FORM: (list **car lis**)

lis
ISA: list
FUNCTION: (argument-in **example1**)

(hold (**fast computers are nice**))

FORM: (text '(**fast computers are nice**))

fast

ISA: atom

FUNCTION: (value-of **example**)

(first **lis**)

FORM: (text **fast**)

When PUPS is presented with an exercise problem to solve in which the goal is to return the first element of a list (A B C) then the structure of the problem is represented in the following way:

goal1

ISA: lisp-code

FUNCTION: (calculate-first. **lis2**)

FORM: ?

lis2

ISA: list

FUNCTION: (hold (A B C))

FORM: ?

Here the form of the final code is unknown and the student, or the PUPS program, has to find a form that will satisfy the functional requirements. In much of his experiments on LISP learning (e.g. Anderson, Farrell and Sauers, 1985), Anderson has used the Winston and Horn (1981) textbook and it is the one used in this case. **example1** above involving car **lis** is used as the source example and is presumed to be available to the student as well as to the PUPS program. If **example1** is the source example and **goal1** is the target, then PUPS generates the following analogy:

function(example1) : form(example1) :: function(goal1) : ?

Anderson et al. go on:

> In solving this analogy, **lis** from **example1** is mapped to **lis2** from **goal1** and the specification (LIST CAR **lis2**) is created for the **goal1** form slot. A similar analogy between **lis** and **lis2** leads to the description (LIST '(A B C)) for the form slot of **lis2**. This constitutes a solution to the problem. (p. 15).

The analogical mechanism described here is somewhat complex. The relations between the function, the form, **car, car-structure, lis, lis1, lis2,** and so on are all specified. Analogical problem solving therefore involves mapping a relational structure from one analogue to another.

Suppose that a student does not have a complete understanding of the syntax used with **car,** as is the case with the student whose verbal protocol appears in Anderson, Farrell, & Sauers (1984). Could such a student successfully solve the problem? PUPS assumes an *a priori* understanding of the set of relations between the function and its form in the example, or at least that the relations between the function and the form are not a source of difficulty. A student who understands the relations between function and form can presumably apply those relations to the target problem (figure 2.14):

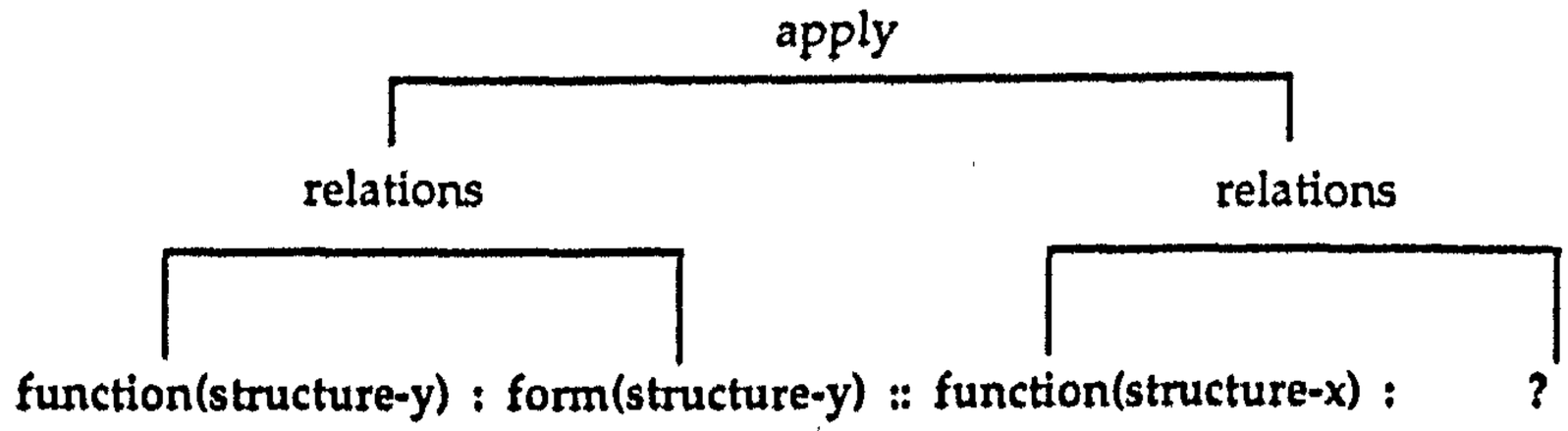


Figure 2.14. Application of a relation between a *function* and associated *form* to an analogous exercise problem in PUPS.

However, there is no need to suppose that analogical reasoning of this kind is required. A student might simply be imitating the form of the earlier example by altering a simple value. All the student needs to know is that the function of **structurey** and the function of **structurex** are the same (they both involve extracting the first element of a list: figure 2.15).

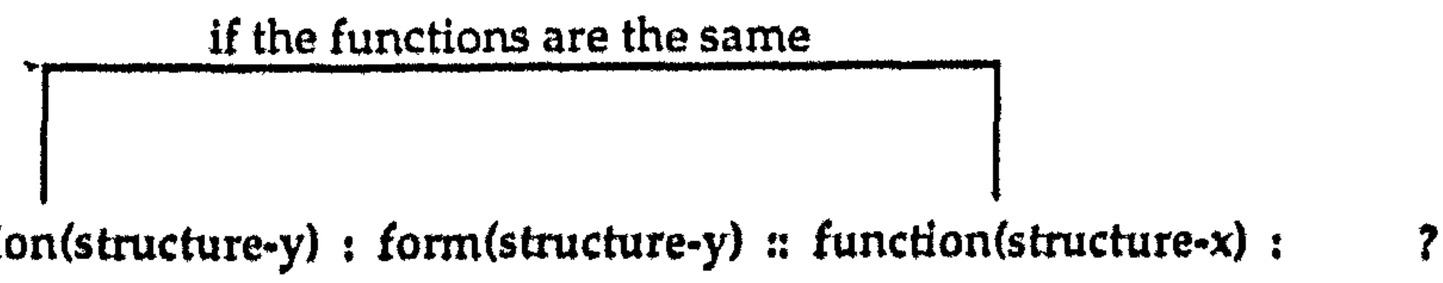


Figure 2.15. Identifying the similarity of two functional requirements in PUPS

That is, **function(structurey)** is like **function(structurex).** If the functions are similar then the forms are similar since they fill the same role, so **form(structurey)** must be like *form(structurex)* (in the same way that 40 mph and 30 mph are the same). All the solver requires to do is to infer that the two values, **(fast computers are nice)** and **(a b c),** play the same roles in both the example and the target problems, so **(a b c)** can be 'plugged in' to the structure given in the source solution. The problem can be solved without necessarily any

understanding of the relationship between the function of the LISP code and the form it takes; the only assumption required is that some relation probably exists (figure 2.16).

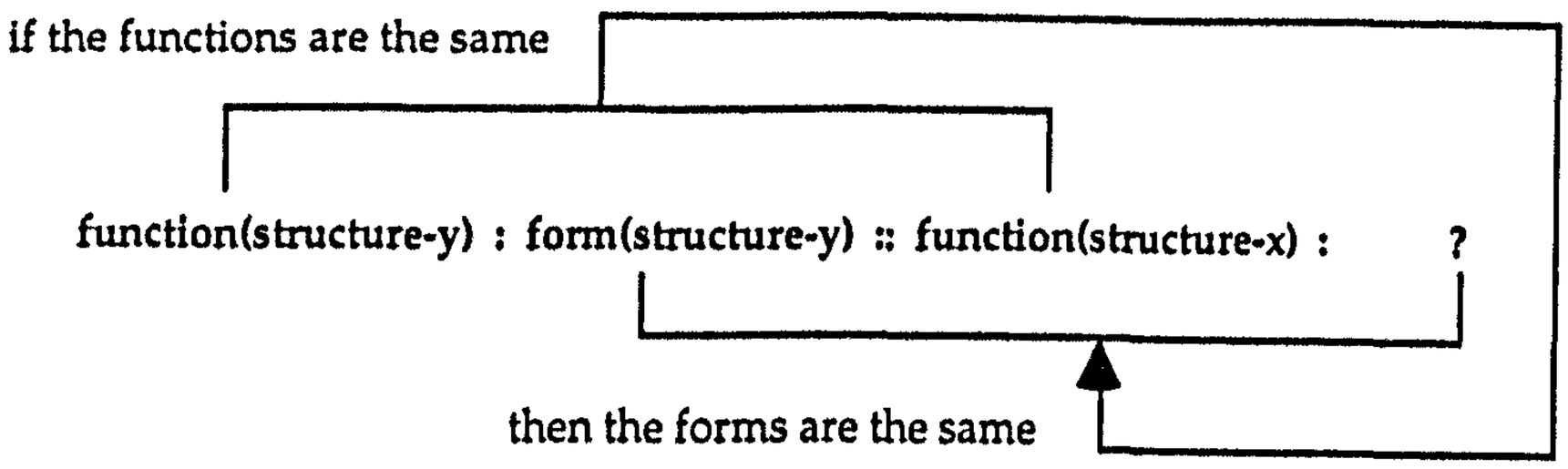


Figure 2.16. Imitation of the form of a previous example to solve an exercise problem in PUPS.

Indeed there is nothing in the PUPS structure to disambiguate the two possible processes, that is: imitation on the one hand and analogical problem solving on the other.

8.2. Summary

Anderson & Thompson (1989) have included the following production rule in PUPS:

IF there is a target structure needing a form serving a particular function
AND there is a model structure containing a form that serves that function
THEN try to map the model form to the target form

The rule contains the advice to 'map the model form' (the problem's surface features) onto another in order to generate a target form which will serve a particular function (achieve the desired goal). There is no requirement (whether intended by Anderson or not) that there should be any *understanding* of the relation between the form and its function, nor that the function is entirely understood. According to VanLehn (1990), understanding is not represented in either ACT* or PUPS (although it can be seen as an emergent property of the representation). The difficulty inherent in this type of problem solving is simply one of mapping corresponding values. This form of problem solving does not involve analogy as it has been defined earlier. It does not require a representation of the problems as a 'problem model.' Although Anderson says that the example is retrieved from LTM, the production rule does not have that condition attached. Indeed one might argue that, if people can remember the example in that much detail then they already have enough information to solve the target problem without reference to the example - that is, they already know what car does. There is no adaptation required of the example other than changing a value.

Anderson and his co-workers have therefore described a process of APS which does not require the solver to understand the base problem. Their description of the process of using an earlier problem is therefore similar to imitation. Their model, on the other hand, relates all the elements of the LISP structure in terms of their type, function and the form each takes. It is a very knowledgeable model - more so, indeed, than any solver needs to be.

9. Conclusion

Human behaviour is an adaptation to the environment. In order to understand why using examples and analogies is pre-eminent in human learning, we need to understand why using them is an optimal adaptive strategy. Generally speaking, the ability to repeat a sequence of behaviours which was successful in achieving one's goals is a useful strategy to employ, and will continue to be successful as long as the environment remains the same. Similarly, copying successful behaviour seen in others is an equally useful strategy. By extension, it is reasonable to suppose that imitating an example presented in a textbook would normally result in a successful solution if the target problem is seen as similar, and the environment (the context) remains the same.

Imitation, therefore, can be understood as a 'rational' adaptive strategy to the environment. Rational, in this sense, does not mean that humans always employ logically correct reasoning. It refers rather to the notion that they will employ an optimal adaptive strategy that is useful in helping achieve their goals (Anderson, 1990; 1991a; 1991b). One of the steps in Anderson's theory of rationality involves achieving a goal by some method that involves the 'least computation.' To the extent that it is possible that two problems are likely to be conceptually similar if they are perceptually similar, the computation involved is mapping the perceptually similar features from one to the other. It is a great deal more computationally demanding to solve a problem by attempting to build a complete problem model than to imitate an earlier one.

The law of rationality makes minimal assumptions about computational limitations. Given a source problem to work from, representing a complex target problem in a new domain is facilitated if it is enough to represent only the surface features in such a way that they correspond to the features of the source. One has also to make the assumption that the perceived similarity in surface features is predictive of similar underlying features.

The basic thesis of a rational approach to using examples has important implications for how an earlier problem is used; that is, that people attempt to derive locally optimal solutions to problems, since computational limitations and limits to our knowledge prevent globally optimal solutions - it is not possible to have an 'ideal problem model' containing all the elements necessary and sufficient to realize a solution plan (Holland et al., 1986) - hence the need for the various heuristics people use in problem solving such as means-end analysis, hill-climbing, and 'if-something-worked-once-try-it-again' whether they know

exactly how it worked or not. It is in this sense that imitation is a rational strategy for problem solving.

Many studies have highlighted the difficulties subjects find in analogical transfer. At the same time analogy can be a powerful tool for helping novices understand new concepts - indeed, analogies are often spontaneously generated by novices to interpret what they read (Clement, 1988; 1989). This appears to be a paradox. However, the paradox disappears if one takes into account the assumed prior knowledge of the novices. When the base domain is well-known and accessible from LTM, then analogies can be drawn from it. Expository analogies work since they appeal to the presumed prior experience of the novice. Analogical reasoning does not work when the novice has little knowledge of the base domain.

With close variants of a source problem imitation will usually be successful (assuming the solver has accessed the relevant source). When solvers attempt distant variants by imitation it usually fails, because aspects of the earlier problem are missing. Solvers can adapt it only if they have a enough domain-specific prior knowledge, which has equipped them with a number of inference rules that they can apply, or if the example gives explicit instructions on how it might be adapted. The literature on analogical transfer shows how rarely this happens.

It takes time for students studying a subject for the first time to develop the 'teleological understanding' that lies behind the procedures they often have to learn (VanLehn, 1990). It has been shown that students have difficulty generating inferences in textbooks since expository texts do not support inferences in the way narrative texts do (Britton, et al., 1990; Britton, et al., 1989). For these reasons students are unlikely to have a representation of an example problem in LTM which they can use or adapt.

The distinction between solving problems by analogy and imitation is not simply one of semantics. It has important repercussions for the design of expository texts. Such texts will be ineffective if they make too many assumptions about what inferences students can derive from examples. Not only should texts provide explanations of the relation between a problem's statement and its solution, but they should also explain how examples can relate to possible variants.

Analogical problem solving is therefore the wrong paradigm in which to understand the processes that people use in solving problems from examples in knowledge-rich domains. If we make minimal assumptions about what a solver knows when faced with a test

problem, we might see more clearly how people develop expertise in these areas, and how the exposition of new information can be improved to enhance learning and transfer.

Chapter 3 DESIGN OF THE INTERPRETATION THEORY

1. Introduction: Requirements for an interpretation theory

In this chapter, I present an interpretation theory that can be used for analysing training and exercise problems as they are presented in textbooks, as well as for analysing the behaviour of students as they attempt to solve those exercise problems. By using a representational scheme that can be adapted to both the text and the behaviour of the solver, we may be in a better position to ascertain the effectiveness of an expository text and to ascertain the processes the solver uses to solve new problems in a formal domain. The theory is based on largely unpublished work done by Hank Kahney of the Open University.

To be effective, an interpretation theory should be built around a model of the learner and a model of the text. In sections 1.1 and 1.2 I shall describe those aspects of a learner and a text that have to be borne in mind when constructing an interpretation theory for text and protocol analysis.

1.1. Constraints provided by the model of the learner

Assumptions about what processes learners employ when using a text to solve an exercise problem constrain any interpretation of texts, tasks and protocols. Novices vary in the strategies and knowledge they bring with them to a new domain. However, in order to apply generally, an interpretation theory should make minimal assumptions about

novices' prior domain-specific knowledge, and about how they use the information presented in expository texts. From the evidence derived from the problem solving literature we can describe some of the characteristics of novices in the following ways:

1) Readers tend not to re-read texts when looking back over them to solve an exercise problem. They tend instead to focus on the examples provided. These can be concrete examples or re-representations of concepts that appear salient, such as diagrams or flow-charts.

2) Faced with an exercise problem, solvers tend to search for a previous example or examples, which they can imitate if it appears similar enough to the current one. The examples they find will be greatly determined by the surface features of those examples; solvers try as far as possible to find an example problem that *appears* to have some similarity to the current problem. They then attempt to apply the procedure instantiated in the example to the exercise problem. To do so they have to identify elements in the source problem that appear to fill the same roles as features in the target. They can then perform the actions that were employed in the earlier example on those mapped elements (assuming they can identify or infer those actions).

3) Expository texts do not support inferences in the way that narrative texts do. Indeed, novices are particularly bad at generating inferences in a new domain (Britton, Van, Gulgoz, & Glynn, 1989; Britton, Van Dusen, Glynn, & Hemphill, 1990).

4) With experience of expository texts, students come to develop a schema for that type of text. Such a schema includes the default assumptions that a solution follows a statement of the problem rather than vice versa, and that a particular section of a textbook will give them enough information to solve problems based on the concepts introduced in that section..

5) Learners vary in what they remember or understand when solving problems based on textbook expositions. Even if they can remember a particular principle in Newtonian mechanics, or the rules for deriving the past participles of French verbs, solvers may not be able to *apply* that principle or rule. This may be because they either do not notice its relevance to a current situation, or because they have not formed inference rules from the principles that will help them solve the current problem or others of the same type.

6) Solvers also vary in the degree to which they can generate 'self-explanations' of new material. To do so they have to have an adequate understanding of the relations between elements in the problem statement and the goals and subgoals in a solution. This

requires the solver to develop some kind of theory of causation. A theory of causation is domain-specific and comes about through applying an already existing (domain-independent) theory of causality to aspects of the new domain (Pazzani, 1991). However, simply to say that better solvers are better able to generate self-explanations (Chi, Bassok, Lewis, Reimann, & Glaser, 1989; Chi & Bassok, 1989) does not tell us how they come to do so.

In some circumstances, for example in learning programming languages such as LISP or PROLOG, there is often little similarity between the surface features of problems. Solvers are therefore obliged to rely on their understanding of the structural similarities between them. To understand the structure, solvers have to rely on the adequacy of the explanation provided in the text. The adequacy of the explanation is a measure of how well the text explains the relation between the examples provided and other problems of the same type.

1.2. Text structure - what has to be represented in an interpretation theory

Texts introduce new concepts in various ways. They may present a verbal explanation, analogies, pictorial representations, or concrete examples. Concrete examples illustrate how a concept is instantiated in a procedure, such as the role of the concept *rate* in amount-time-rate problems. As students progress through textbooks or training manuals the writer has to rely on the reader having learned concepts introduced earlier in the book, so that, when new concepts form part of an example or form a subprocedure, the writer has to assume that the reader understands the rest of the procedure (which is not explained in the current section of the book). The reader is then often obliged to make *reinstatement inferences* (Kintsch & Van Dijk, 1978) or engage in a *reinstatement search* to understand concepts that were introduced in an earlier part of the textbook.

To make explanations clearer, texts may present concepts and examples at different levels of abstraction; from concrete examples to abstract or pictorial representations or analogies. The writer assumes that the solver is able to abstract out the structure of the intermediate representation and map it onto the example problem. Intermediate representations only achieve their intended effect if the reader understands them well enough to make this mapping.

The set of operations applying sequentially to elements in the problem statement constitutes a procedure that applies to a class of problems, although there can be variations in the procedure between problems. The more variations there are between

problems, the more 'distant' they are. A large degree of variation leads to a hierarchy of problem types. Figure 3.1 shows such a hierarchy for algebra 'rate' problems.

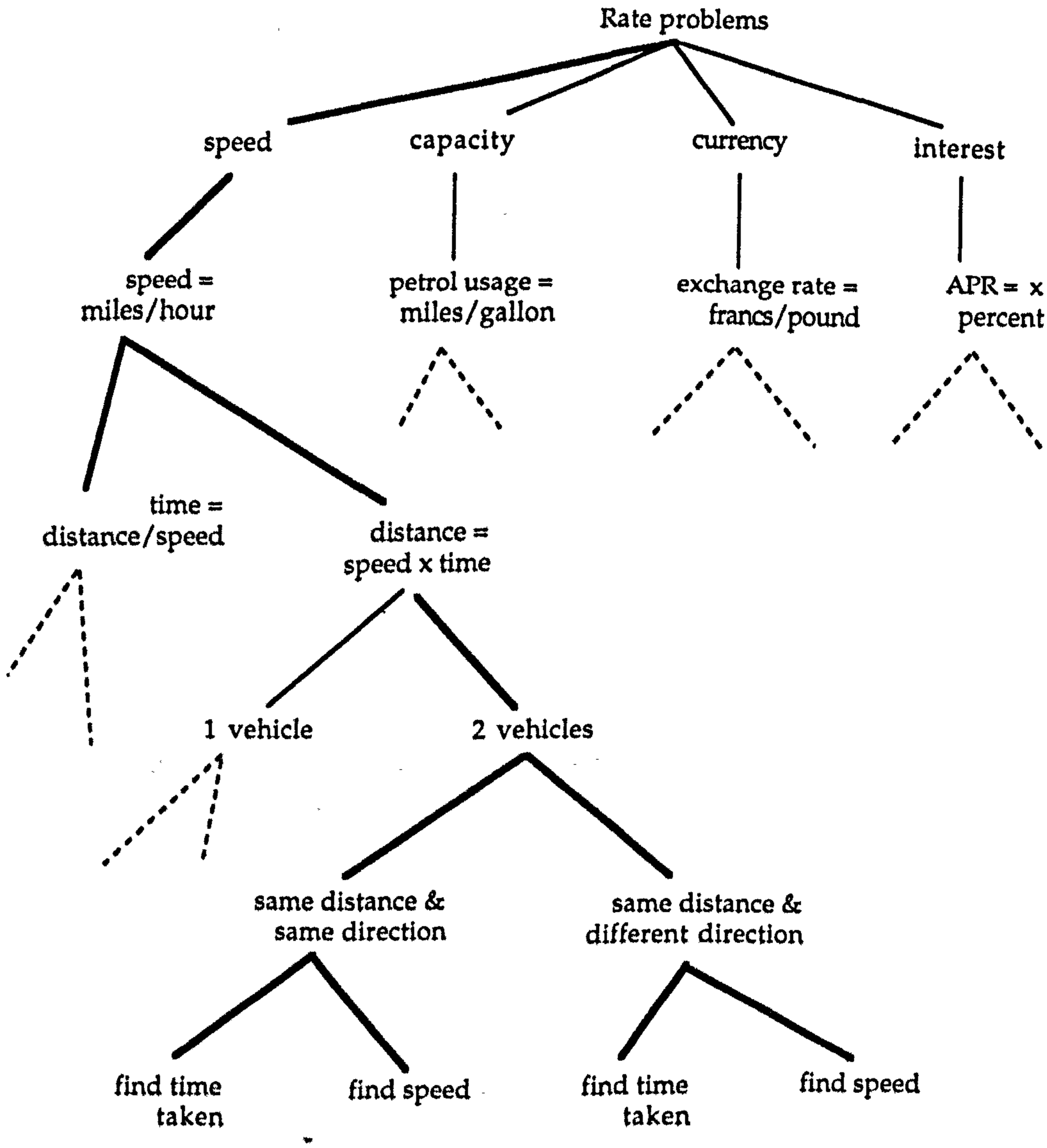


Figure 3.1. A hierarchy of rate problems.

The four rate problems (speed, capacity, currency, interest) involve the same underlying equation ($a = b \times c$) but different surface features. At the bottom of the hierarchy are *distance = speed* x *time* problems involving two vehicles. Close variants would be two problems in which the time is unknown and has to be found. More distant variants would involve an example in which the time is unknown, and a test problem in which the speed is unknown. More distant still would be problems from a different part of the hierarchy; an example problem in which vehicles travel in the same direction is a distant variant of a problem in which the vehicles travel in opposite directions.

Texts also, at times, provide explanations of what a problem statement or a solution *means*. This would occur where, for example, the SOLO programming manual explains how a solution program is evaluated. On page 38 of the SOLO manual there is an example showing how to define new procedures. The procedure is 'TO GRUMBLE' containing three PRINT statements. The manual then goes on to explain what happens when this procedure is activated. Explanations of the relation between elements in a problem statement and a solution, and explanations of what a solution means are highly interconnected. Nevertheless, texts often show how a solution is evaluated without providing a *full* explanation of the relation between statement and solution.

1.3. How the interpretation theory can represent texts and the behaviour of solvers

The goal of the interpretation theory is to describe:
a) the relation or set of relations between a statement of a problem and its solution. This includes, where necessary, a breakdown of a problem into its goal-subgoal structure;
b) the relations that exist between a series of problems; that is, information about what makes a series of problems representative of a category;
c) the relations between example problems and exercise problems.

If the theory can adequately describe the examples and the relations between them, then it can also pinpoint where those relations do *not* exist. In other words, where there are no explicit relations given, the reader is obliged to make inferences. Similarly, when an exercise problem contains subgoals that do not figure in the immediately preceding section, the solver may be required to make reinstatement inferences. Through an examination of the relation between exercise and example problems, an interpretation theory can illustrate the 'path' that an ideal solver has to take through the text when solving a problem. This path should allow us to see where the path is 'blocked' - that is, where there is no previous corresponding element or operator in the text - and therefore where the solver has to make inferences. In so doing the theory can highlight the usefulness of the text in helping the solver.

Furthermore, by analysing the path the text traces with the path a solver *actually* follows, we can see how the two compare. If the solver does not follow the implicit path in the text, then we can get some measure of the 'impenetrability' of the text, that is, the degree to which the text is not sufficiently explicit. The theory should allow us to determine what assumptions are built into the text about what the solver is expected to know, and should also be capable of illustrating what aspects of a text or problem the solver is trying to map and what concepts he or she is trying to understand.

1.4. The basis of the interpretation theory

There already exists a methodology for analysing how solvers establish relations between analogues in the literature on proportional analogies. The interpretation theory described in section 2 uses the proportional analogy framework as a basis for examining the mappings that are given in a text and those that the solver makes.

1.4.1. The proportional analogy framework

The analogical mapping technique described here is based on the theories of problem solving (by, for example, Spearman, 1923 and Sternberg, 1977) on proportional analogies of the type A : B :: C : ?D. For example, the following problem:

Lawyer : client :: doctor : ?WHAT?

should be understood as:

Lawyer is to client as doctor is to ?WHAT?

The task for the solver is to infer some relation between *lawyer* and *client* which can then be applied to *doctor* to yield an answer, for example *patient*.

Using this framework, it is possible to analyse the structure of texts in terms of the training examples they provide and the test problems learners are expected to solve. It fulfils the requirements of an interpretation theory outlined above in that it is capable of illustrating the mapping processes which 'perfect' students would use if they refer back to previous examples in a text in order to solve an exercise problem; and it can be used to portray the actual mapping processes of any particular solver. It can show how much information is given in the text immediately preceding a test problem.

The technique can also be used to analyse the structure of complex problems. Such problems can be broken down to represent the goal-subgoal structure of the problem as well as any constraints which are specifically mentioned in a problem statement. Breaking a problem down in this way allows us to see how a problem is structured and what kind of conceptualization of the problem is required. As a result the theory provides a metric of transfer. That is, from an examination of the inferences a student has to make to use a source to solve a target problem, we can derive some measure of the 'distance' of one problem from another. Close variants of example problems require the solver to make few additional inferences; they can be solved simply by imitating the procedure provided in the source, which involves replacing the surface features that fill the same roles in both problems. Distant variants, on the other hand, will require the solver to make inferences or to manipulate their representation of an earlier problem to solve the target.

Since the technique can be used to look at the relation between target and source problems as a whole as well as the relation between the subgoal structure of a target and earlier source problem, the analysis can provide a representation of problem solving from examples at different 'grain' sizes. An overview of a section of a textbook in a specific domain can show how the section is structured in terms of the examples provided. At a finer grain, the paths necessary to solve a test problem presented in the textbook can be drawn, indicating where a novice would have to look when using earlier examples to solve a current problem.

2. Proportional analogies

Proportional analogies of the form A : B :: C : D are known as 4-term analogies. The A and B terms are known as the *domain* and the C and D terms are known as the *range*. the number of terms can be increased or decreased. The analogies described by Sternberg (e.g. Sternberg, 1977) were three-term analogies such as *red : stop :: green : ?WHAT*.

It is also possible to complete two-term analogies where there is no choice offered, for example:

? : kitten :: dog : ?WHAT

Here the A and D terms are missing. By looking for a relation between 'kitten' and 'dog' a solver might establish that the relation is one of relative *size*. 'Kitten' is small and 'dog' is large in comparison. However, this is still insufficient to solve the analogy. The analogizer has to infer further that 'kitten' and 'dog' belong to different categories; 'kitten' can be classed as a type of 'cat' and 'dog' is itself a category name. Furthermore, the relation of 'relative size' has to apply in one direction in the domain and in the opposite direction in the range. That is, the relation has to go from small to large in the domain to generate 'cat' and from large to small in the range to generate 'puppy'.

This kind of analogizing, although possible, is obviously more difficult than if the A term were available for study. Nevertheless, this is the situation that some solvers sometimes put themselves in when they look back to an example solution (equivalent to the B term) and use it to solve an exercise problem (equivalent to the C term) and ignore the earlier problem statement (the A term).

2.1. Sternberg's Componential Theory of Analogical Reasoning

In Sternberg's componential theory of analogical reasoning: 'an analogy exists when there is a higher-order relation of equivalence or near-equivalence between two lower-order relations (*A is to B* and *C is to D)*' (p. 134). Sternberg proposes a number of processes that occur when a solver attempts to form an analogy. Some of these are:

1. *encoding*

Encoding an item involves the activation of a set of semantic features associated with a concept. These features might include superordinate, subordinate, property and others.

2. *attribute comparison*

A comparison process operates between the A and B terms whereby the solver infers some relation between them.

3. *mapping*

Mapping is another attribute comparison process, this time involving the search for correspondences between the A and C terms.

4. *application*

This involves applying a rule which can apply the relationship inferred between the A and B terms to the mapped features of the C term.

The A and B terms are first encoded. A relation (X in figure 3.2) is inferred if the solver finds related attributes among the features of the two terms.

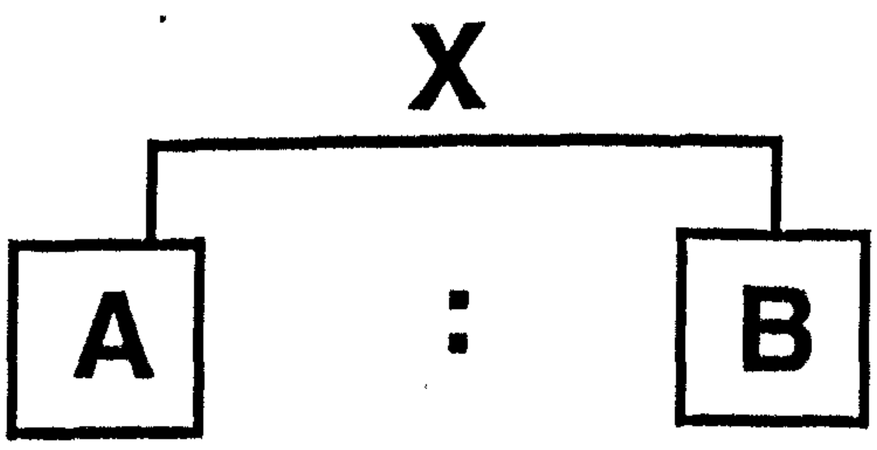


Figure 3.2 X refers to the lower-order relation between the two terms (A and B) of the *domain* of the analogy.

The next stage is to find the correspondences between the A and C terms (figure 3.3):

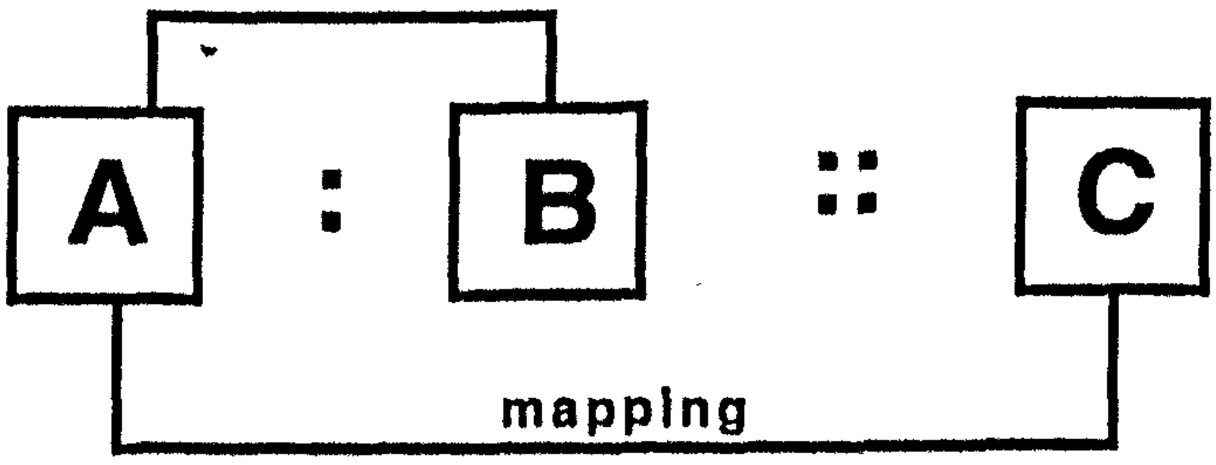


Figure 3.3 Mapping the A and C terms of an analogy

The higher-order relation can only be applied once the relevant attributes and values in the A and B terms have been identified and once there has been a mapping between the A and C terms to discover the relationships between them. Application of the relation

between the terms of the domain can then be applied by generating a rule (Y) which applies the inference relationships between the A and B terms to corresponding values in the C term, thereby forming an ideal solution D. These processes are summed up in figure 3.4.

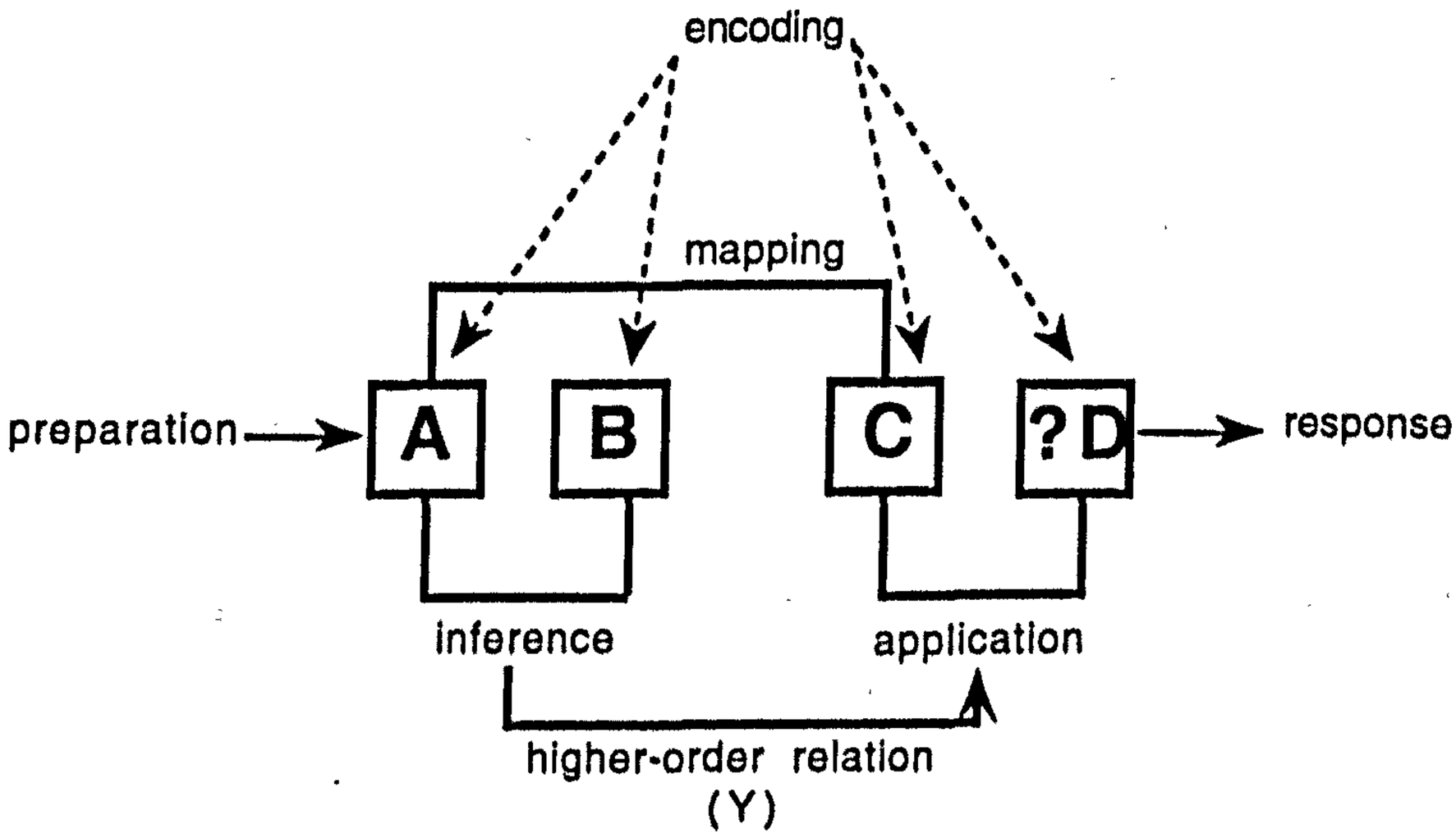


Figure 3.4 Sternberg's componential theory of analogical reasoning. The figure includes the component processes of *encoding*, *inference*, *mapping*, and *application* and the *control* components of *preparation* and *response*. (adapted from Sternberg, 1977, p.136).

2.2. Variations on Sternberg's theory

2.2.1 Heller's model

Heller (1979) proposed a variation on Sternberg's model in which the sequence of the component processes differs from that of Sternberg. The model (which concentrates principally on verbal analogies) takes account of the variation solvers bring with them in terms of their prior knowledge and reasoning strategies as well as variations in the constraints and characteristics of the particular tasks they are required to do. It also emphasizes the interaction between these. In analogical reasoning tasks, where the solver is given a choice of solutions (D1, D2, etc.), the solver may infer a relation between the A and B terms (the 'relationship identification process'), then infer a relation between C and D1, or C and D2 and *compare* this with the relation between the A and B terms (the 'relational comparison process'). Where there is a choice of solutions the solver has to 'align' two or more C-D relations with the A-B relation in order to decide on a solution (the 'relative match comparison process'). Thus her Conceptual Model replaces Sternberg's mapping process with a comparison process (figure 3.5).

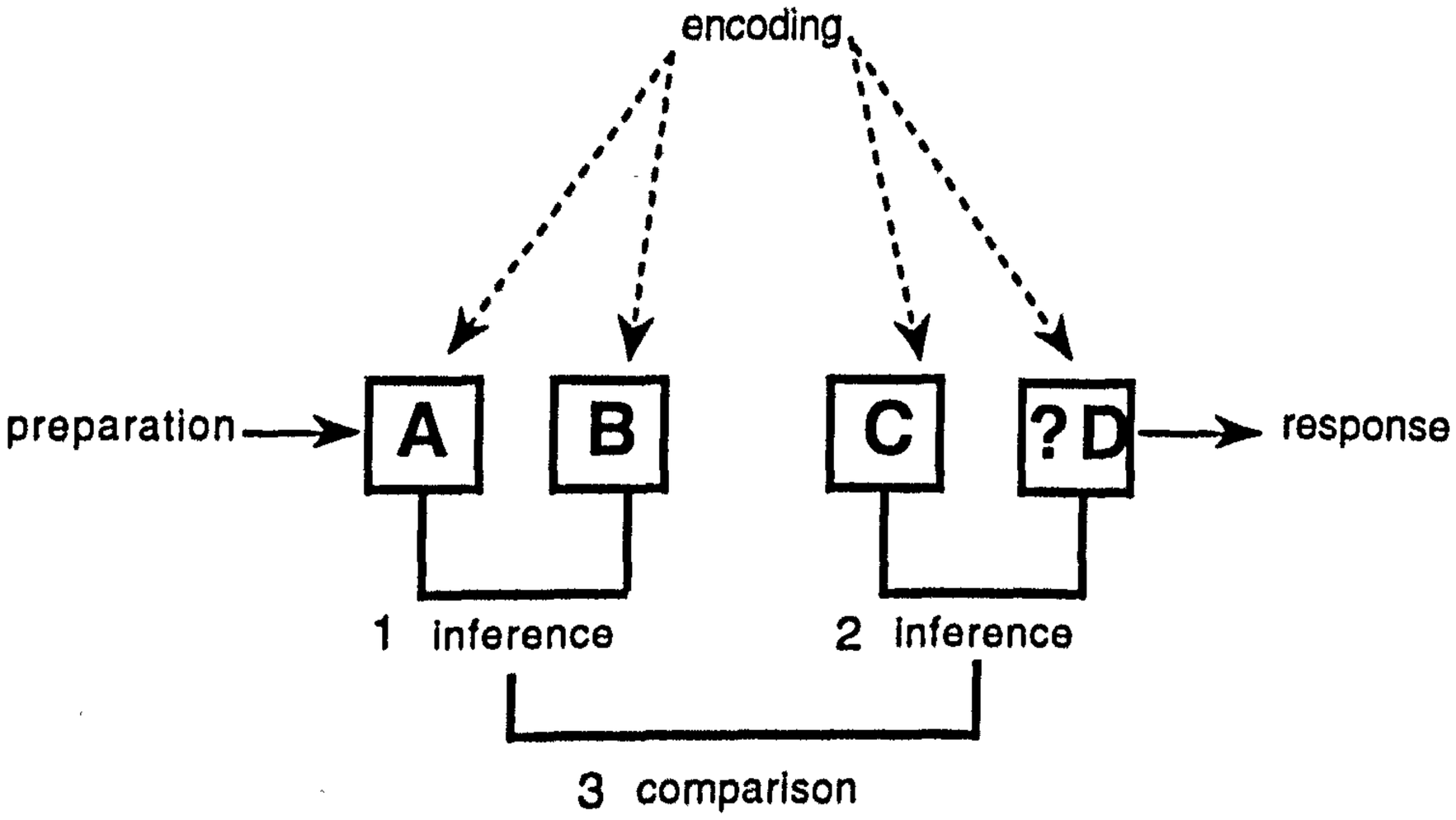


Figure 3.5. Heller's *conceptual model* of analogical reasoning (figure adapted from Kahney 1993).

Heller also argues that, when solvers fail to infer the relation between the A and B terms, good solvers may look at the C and D terms and infer a relation between them and look back at the A and B terms to see if that helps them find a relation between these. Alternatively, they may hypothesize some relation between the A and B terms, which is later revised in the light of the relation inferred between the C and D terms. In fact, there is no reason why a solver might not engage in a number of possible comparison processes between the A and B terms, the A and C terms, the B and D terms, the C and D terms, and so on. In this Interactive Model, the solver may make any number of such inferences and comparisons in order to find a justifiable solution. Some of Heller's subjects also made comparisons which violated the structure of these problems. For example, they made irrelevant inferences and comparisons between the A and D terms, the B and C terms, the A-B-C terms and so on (see Kahney, 1993, for a fuller discussion of Heller's model.)

2.2.2. Grudin's model

Grudin (1980) pointed out that the mapping from A to C may in some cases be more useful than that from A to B in inferring a relation which can be applied to the B-term to generate a solution. For example, in the analogy *diamond : elm :: gem : tree* it was found that inferring the relation between the A and C terms and applying that relation to the B term to derive the D term was more likely to produce a solution than inferring the relation between the A and B terms when the D-term was missing. A relation can be readily inferred between diamond and gem but not between diamond and elm. In this case the relationship would be hyponymic (relating subordinate and superordinate categories).

Grudin states that, in Sternberg's model, the A : B relation is applied to the C term *despite* the mapping of the A and C terms. This can be seen in figure 3.6 which compares what he calls the 'standard theory' with Sternberg's and his own.

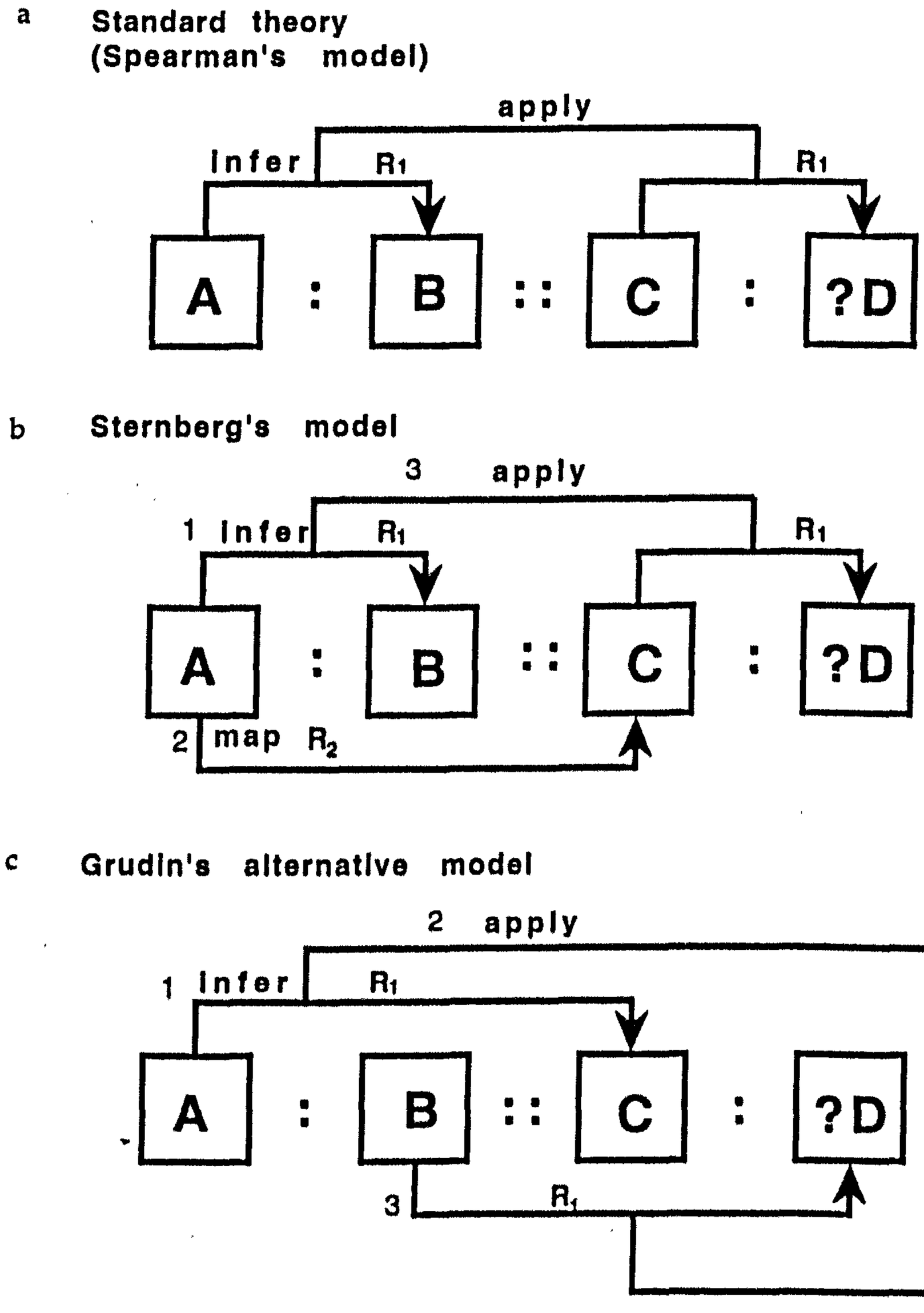


Figure 3.6. Comparison of Spearman's, Sternberg's and Grudin's model of proportional analogical reasoning (based on Grudin, 1980). Sternberg's model includes a mapping process which is absent from the other two models. 'R' denotes an inferred relation.

Sternberg, in his exploration of reversible analogies, suggested that a subject first tries to infer an A-B relation and a mapping from A to C. Only if this fails does the subject infer an A-C relation and map A-B, but it is still the A-B relation which is applied. Grudin,

however, argues that there is not always a need for a mapping process. When subjects fail to infer a relation between the A and B terms, they try to infer one between the A and C terms (step 1 in figure 3.6 c). That relation is then applied to the B term directly (step 2). Given, as Grudin points out, the flexibility exhibited by people solving problems, and the range of difficulty of analogy problems of this type, we might reasonably expect a range of problem solving processes, so that Sternberg's, Heller's and Grudin's models can be seen as complementary.

2.3. Strategies in analogical reasoning tasks

Although Sternberg's model may be considered the 'paradigm' of analogical reasoning strategies, solvers may successfully use a variety of strategies in solving proportional analogies (particularly verbal analogies). Some are more likely to lead to success than others. Table 3.1 lists a number of possible processes. The left-hand side shows the relations that have to be inferred (A?B, however, refers to an as yet unknown relation between the A and B terms). The right-hand side of the table describes the process that applies to the inferred relations. Thus a solver might infer a relation between A and B, and C and D and either *compare* the inferred relations or use the C-D relation to *reformulate* the A-B relation.

Table 3.1
Analogical reasoning strategies

infer relation	*process*
A-B	apply the inferred relation to C to generate D
A-B & A-C	constrain/reformulate A-B relation by finding the correspondences between the A and C terms and apply to C to generate D
A-B & C-D	compare the two relations
A?B & C-D	use C-D relation to generate A-B relation
A-B & C-D	use C-D relation to reformulate A-B relation
C-D1 & C-D2	use relations between C and 2 alternative solutions to establish possible A-B relations and compare
A-C	apply to B to generate D

Table 3.1. Analogical reasoning strategies.

Although Table 3.1 shows a number of possible analogical reasoning strategies they are not exhaustive. Other strategies can be used but they may not necessarily lead to successful solutions. Indeed there are many possible relations, interrelations and mappings

that can be made but which violate the constraints of the particular task (see Heller, 1979, p. 51, for a list).

3. Adapting proportional analogies to describe the structure of texts and problems

In order to identify and discriminate between example problem statements and their solutions, and test problem statements and solutions, the interpretation theory uses the proportional analogy framework. The A term represents a textbook example (source) problem statement, the B term is the example solution, the C term is the test (target) problem statement and the ?D term is the solution the solver is attempting to discover (figure 3.7).

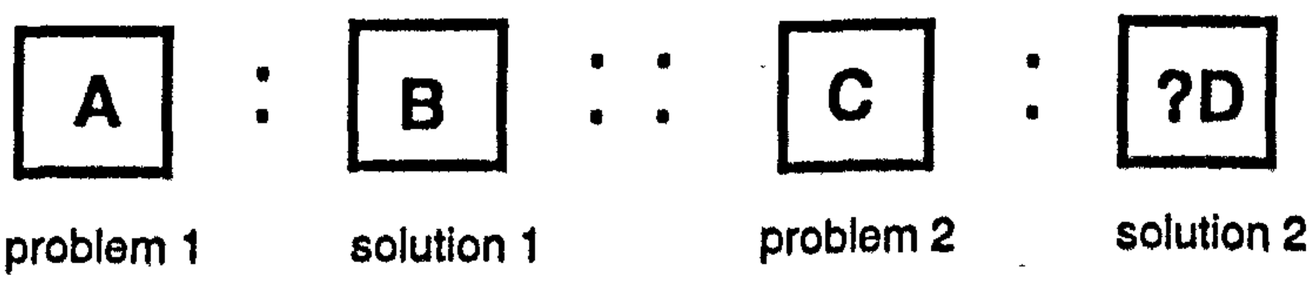


Figure 3.7 Training problems and exercise problems within a proportional analogy framework

An example would be:

Fortress • Fortress :: Radiation • Radiation
Problem • Solution :: Problem • Solution

Used as a tool for text analysis, this framework allows us to break down a section of a scientific textbook into its constituent parts represented by the four term analogy, and shows how these parts are related to each other. As a tool for task analysis, it provides a means of analysing a complex exemplar problem in terms of the relationships between the problem givens and the goals and the subgoals required for its solution. Representing test problems by C and D terms provides a means whereby the solution pathways which have to be taken to solve them can be clearly mapped out. That is, links can be made between a test problem and those parts of a source example that relate directly to the solution of the test problem. This in turn allows us to see what inferences, analogies or adaptations a solver is required to make, given the structure of the text and the goal-subgoal structure of the examples provided, and therefore what aspects of a problem are presumed by the writer of the textbook to be already represented in a solver's memory.

A solver will seek a potentially relevant source example which is noticed or retrieved through some perceived similarity to the current target problem statement. In the terms of proportional analogy this would yield figure 3.8 where the target C term, either by hint or some other form of reminding, recalls the source A term.

Figure 3.8. Accessing previous example.

In situations where 'ideal' students are learning by reference to an example, and where they cannot remember a specific relevant example, one might reasonably expect that they would they would try to find one in the textbook and go on to discover the relation or transition function between the initial state (A term) and the solution (B term) of the source problem (figure 3.9) at a level of abstraction sufficient to allow that relation to be applied to the C term.

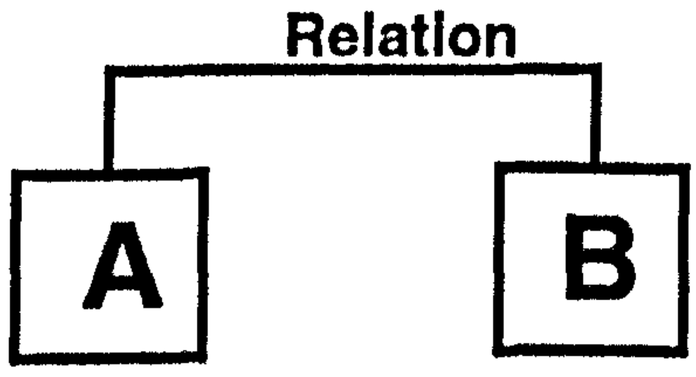


Figure 3.9. Representation of the relation between an example problem statement and its solution.

Elements in the A term are mapped onto the C term and the higher-order mapping relation established between the A and B terms is applied to the mapped elements in the C term to yield the D term. These processes are represented in figure 3.10[1] which represents the ideal to which expository texts should aspire. It also represents the behaviour of the ideal student:

[1] *Not only does figure 3.10 represent some of the component processes assumed to be involved in solving problems using a previous example, but it also represents the student's schema of how examples provided in a text can be used to solve a current problem. In a text one would expect a problem to be stated and then the solution to be presented along with an explanation of how one relates to the other. A solver would expect that relation to apply to elements in the target problem. However, it often happens that the ordering of worked examples does not follow such a canonical form.*

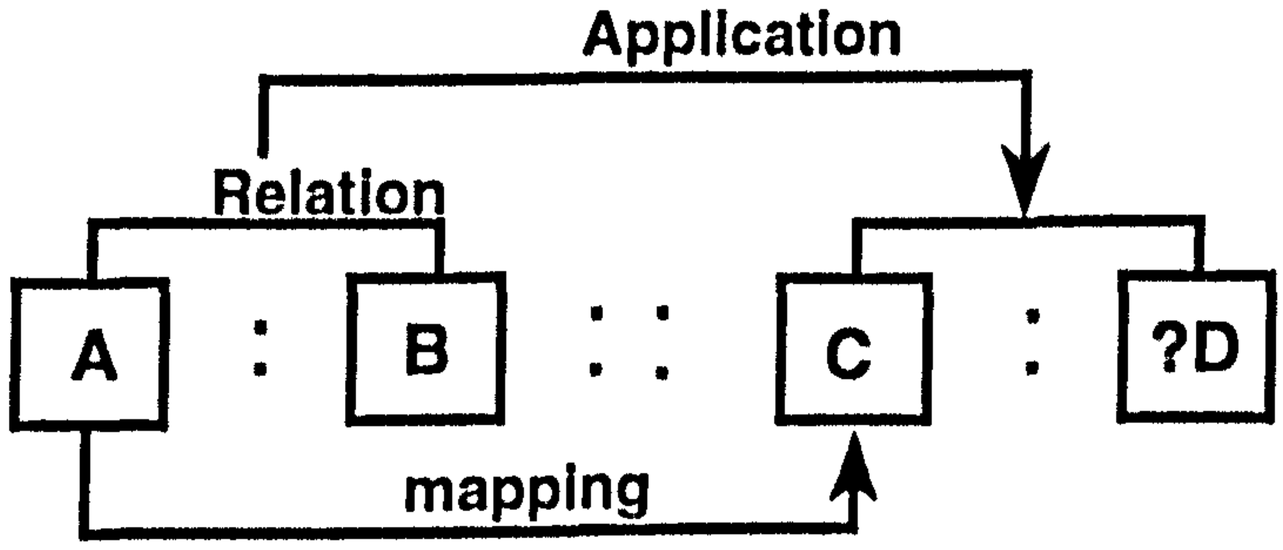


Figure 3.10. Component processes of analogical problem solving.

3.1. Distinctions between the interpretation theory and proportional analogy

According to Sternberg's analysis of proportional analogy, encoding of single terms involves a search through a semantic net for attributes that can link the A and B terms. Encoding in problem solving is a more complex process involving text processing of the problem statement. In this case the literature on text understanding becomes more appropriate in understanding how a problem statement, or textbase, is encoded (Kintsch, 1986; Kintsch & Van Dijk, 1978; Van Dijk & Kintsch, 1983). Indeed, Kintsch (1988) argues that, in simple arithmetic word problems, the generation of inferences from the text alone may generate a solution to the stated problem without the solver having to engage in any actual problem solving activity. In more complex problems the encoding of the textbase is the necessary first step in generating some kind of problem model.

According to Egan & Greeno (1974) analogical reasoning, series completion, problem solving and concept formation all require a search for relations among elements resulting in new interconnections between the nodes of a network structure. In solving problems from examples the discovery of a relation between terms does not necessarily involve a search through a semantic network for attributes that relate concepts. Rather, writers provide examples in order to establish the awareness in readers that a relation between concepts exists. The nature of such a relation often comes about by inductive processes on the part of learners that allow them to associate two disparate concepts (depending on the extent to which the example problem has been elaborated). Such inductive processes may include, for example, applying a theory of causality in trying to understand such relations. However, the learner may not always understand the relation.

Where the A and B terms refer to a problem statement and solution in a textbook, the relation between the problem givens and the solution is already stated or explained (more or less well). In worked examples, then, the relation between a problem statement and its

solution should be *provided* and the task for the solver is to *apply* that relation to a novel problem. In such cases, once the correspondences between an exercise problem and a training problem have been established, it is primarily the higher-order mappings which have to be made. That is, the solver has to extract the rule or procedure instantiated in the example problem and apply it to the new problem. This process is one of generalization. It is this generalization process that allows the solver to abstract out the relevant solution schema, and to begin to induce its range of applicability.

Evaluation of the response in problem solving, a process that Sternberg believes takes place with proportional analogies with multiple-choice responses, depends on the type of problem given. Some types of problem give the goal state in the statement of the problem (e.g. Duncker's Radiation problem). In this case the goal is either achieved or it isn't. In algebra word problems the goal state is unknown or is only vaguely specified and evaluation may be difficult. In computer programming problems a solution is often evaluated by the computer itself where a solution program either functions as it is supposed to according to the problem or it fails to do so or generates an error message.

4. The form of the interpretation theory

In this section the interpretation theory is presented in detail. To show how the relations between the four terms of the proportional analogy framework can be applied to analogical problem solving, I will examine first of all one of the paradigms of the analogical problem solving literature: the studies of Gick and Holyoak (1980; 1983).

4.1. The Fortress and Radiation problems

In their studies of analogical transfer Gick and Holyoak used two isomorphic ill-structured problems known as the Fortress Problem and the Radiation Problem (from Duncker, 1945). Using various experimental manipulations, Gick and Holyoak attempted to establish the conditions under which transfer would occur or fail to occur. The relations between the surface features and the underlying structure of the two stories will be examined to see where they are similar and where they are different. (The texts are taken from Gick and Holyoak, 1983.)

The Fortress Problem

A small country was ruled from a strong fortress by a dictator. The fortress was situated in the middle of the country, surrounded by farms and villages. Many roads led to the fortress through the countryside. A rebel general vowed to capture the fortress. The general knew that an attack by his entire army would capture the fortress. He gathered

his army at the head of one of the roads, ready to launch a full-scale direct attack. However, the general then learned that the dictator had planted mines on each of the roads. The mines were set so that small bodies of men could pass over them safely, since the dictator needed to move his troops and workers to and from the fortress. However, any large force would detonate the mines. Not only would this blow up the road, but it would also destroy many neighbouring villages. It therefore seemed impossible to capture the fortress.

However, the general devised a simple plan. He divided his army into small groups and dispatched each group to the head of a different road. When all was ready he gave the signal and each group marched down a different road. Each group continued down its road to the fortress so that the entire army arrived together at the same time. In this way, the general captured the fortress and overthrew the dictator.

The Radiation Problem

Suppose you are a doctor faced with a patient who has a malignant tumour in his stomach. It is impossible to operate on the patient, but unless the tumour is destroyed the patient will die. There is a kind of ray that can be used to destroy the tumour. If the rays reach the tumour all at once at a sufficiently high intensity, the tumour will be destroyed. Unfortunately at this intensity the healthy tissue that the rays pass through will also be destroyed. At lower intensities the rays are harmless to healthy tissue, but they will not affect the tumour either. What type of procedure might be used to destroy the tumour with the rays, and at the same time avoid destroying the healthy tissue?

In Gick and Holyoak's Fortress problem the solution involves a division of a large force (an army) and its convergence on a target (the fortress) as represented in figure 3.11. However, each of these main relations (divide and converge) themselves involve a number of operations and transitions which also need to be imported into, and often transformed in, the new domain. These unspecified operations and transitions are represented by the 'black boxes' in figure 3.11.

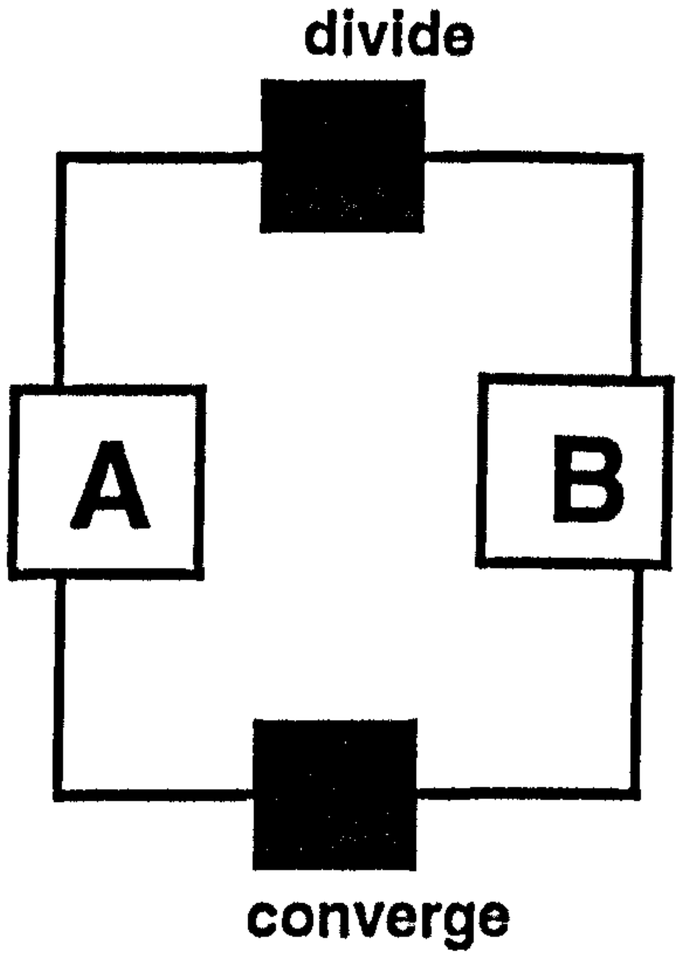


Figure 3.11. 'Divide' and 'converge' relations in Gick and Holyoak's Fortress problem.

When solvers use the Fortress problem to solve the Radiation problem elements in the A term are mapped onto the C term and the higher-order mapping relations established between the A and B terms are applied to the mapped elements in the C term to yield the D term. Applying the earlier solution, according to Gick and Holyoak, involves representing the causal relations between the A and B terms of the source problem at an appropriate level of abstraction, in this case 'divide' and 'converge'. Solving a *specific* problem means finding the *specific* operations and transitions represented by the black boxes in figure 3.11. In the Fortress problem, the general is acting under the constraint of protecting his army, whereas the surgeon in the Radiation problem does not have to protect his ray-machine. The singular entity (army) can be divided up into several entities (small groups) but a ray-machine cannot. Instead, the surgeon has to find lots more machines. In the Fortress problem, the groups have to be moved and positioned at the head of the roads leading from the fortress. There are no fixed pathways leading to the tumour in the Radiation problem. The groups attack the fortress by simultaneously converging on it along the roads, but *capturing* (a fortress) is not the same as *destroying* (a tumour). Figure 3.12 elaborates on figure 3.11 by including these relations.

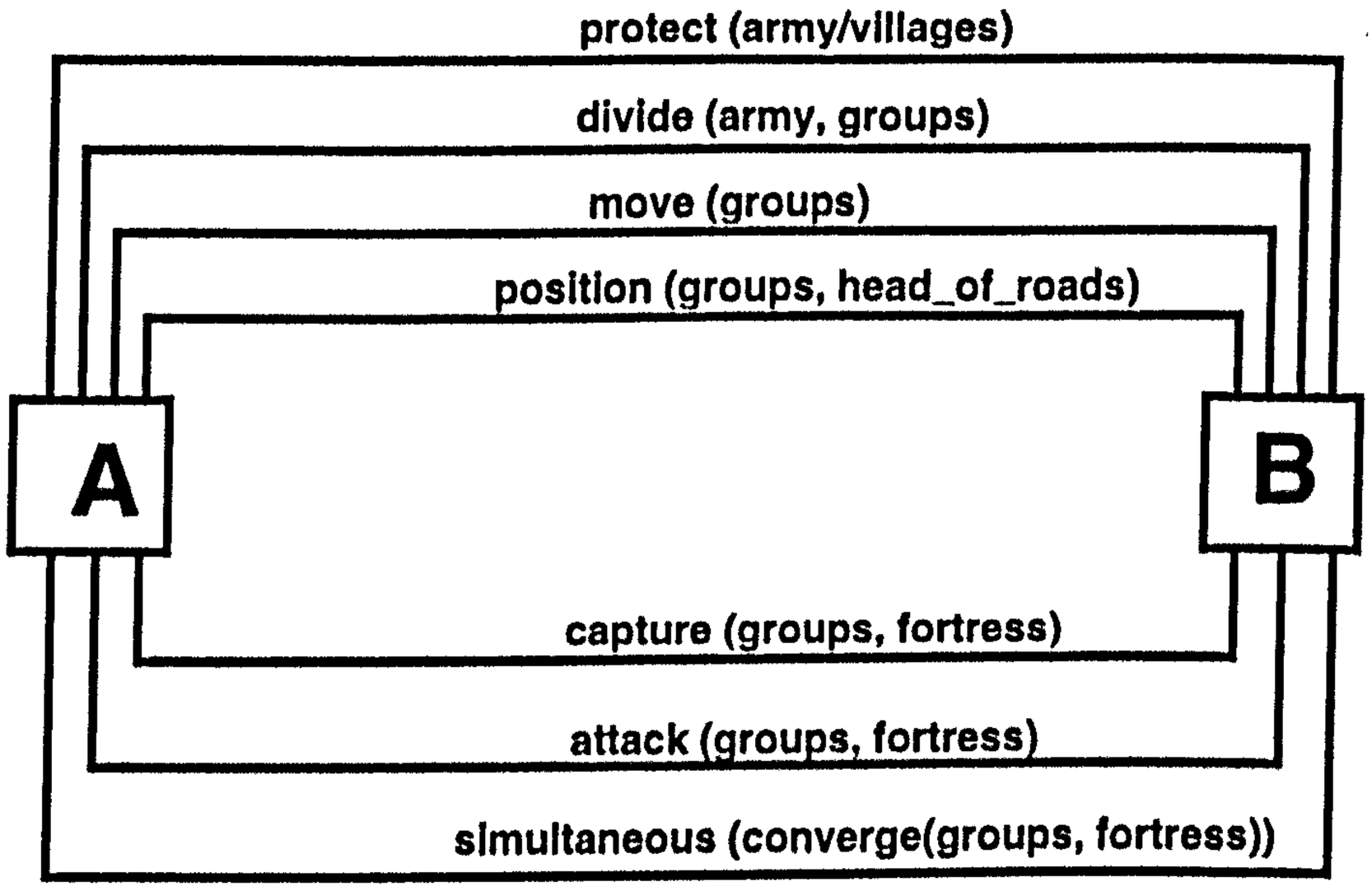


Figure 3.12. Specific relations in the Fortress problem.

The rebel general is operating under two main constraints when devising a solution plan: *protect(army)* and *protect(farms & villages)*. The predicates in figure 3.12 represent the operators (divide, move, position, etc.) that should be applied to the 'objects' (A1, A2, etc. in figure 3.13) in the Fortress problem in order to generate a solution. Figure 3.14 lists the objects in the Radiation problem.

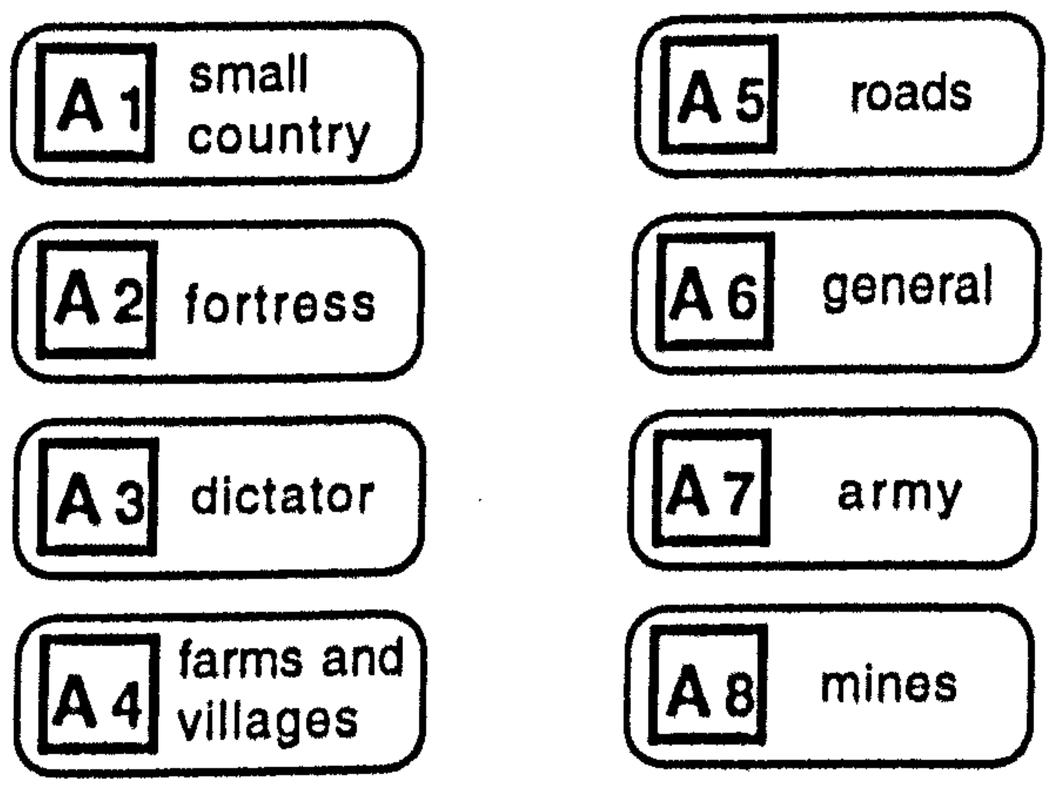


Figure 3.13. The objects of the Fortress problem.

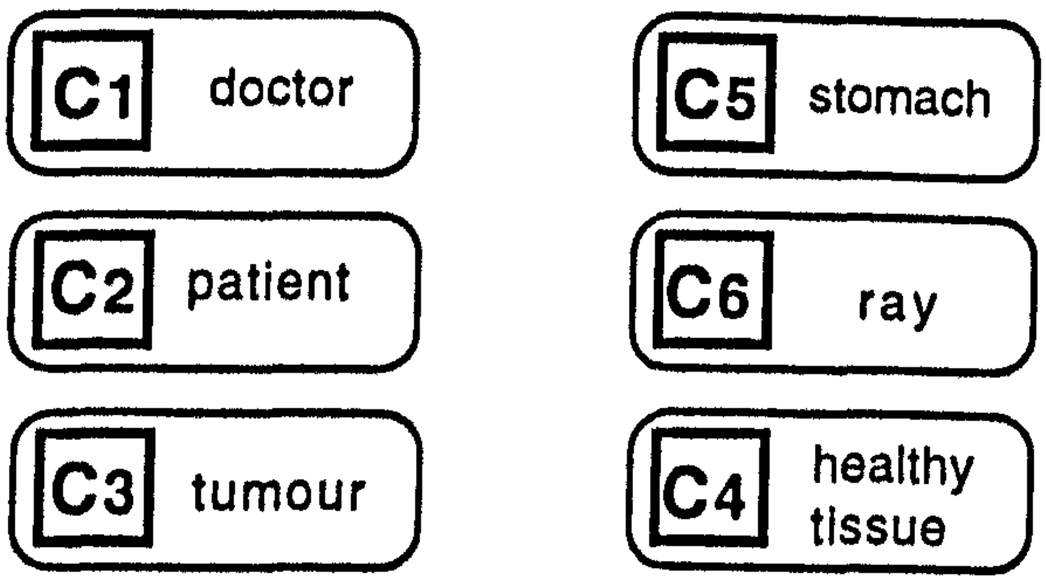


Figure 3.14. The objects of the Radiation problem

Part of the difficulty in solving the Radiation problem from the Fortress problem is making the necessary mappings between the two problems and then making the necessary manipulations to the representation the solver has of the Fortress problem so that the target can be solved. The mappings are not straightforward. In the Fortress problem the general has to protect the army and the villages. In the Radiation problem the surgeon has to protect the healthy tissue. 'General' can be mapped directly onto 'doctor' (A_6 maps directly onto C_1 in figure 3.15). However, there are two possible mappings for 'healthy tissue': 'army' and 'villages' (A_4 and A_7 both map onto C_6 in figure 3.15: the double arrows in the figures represent those mappings).

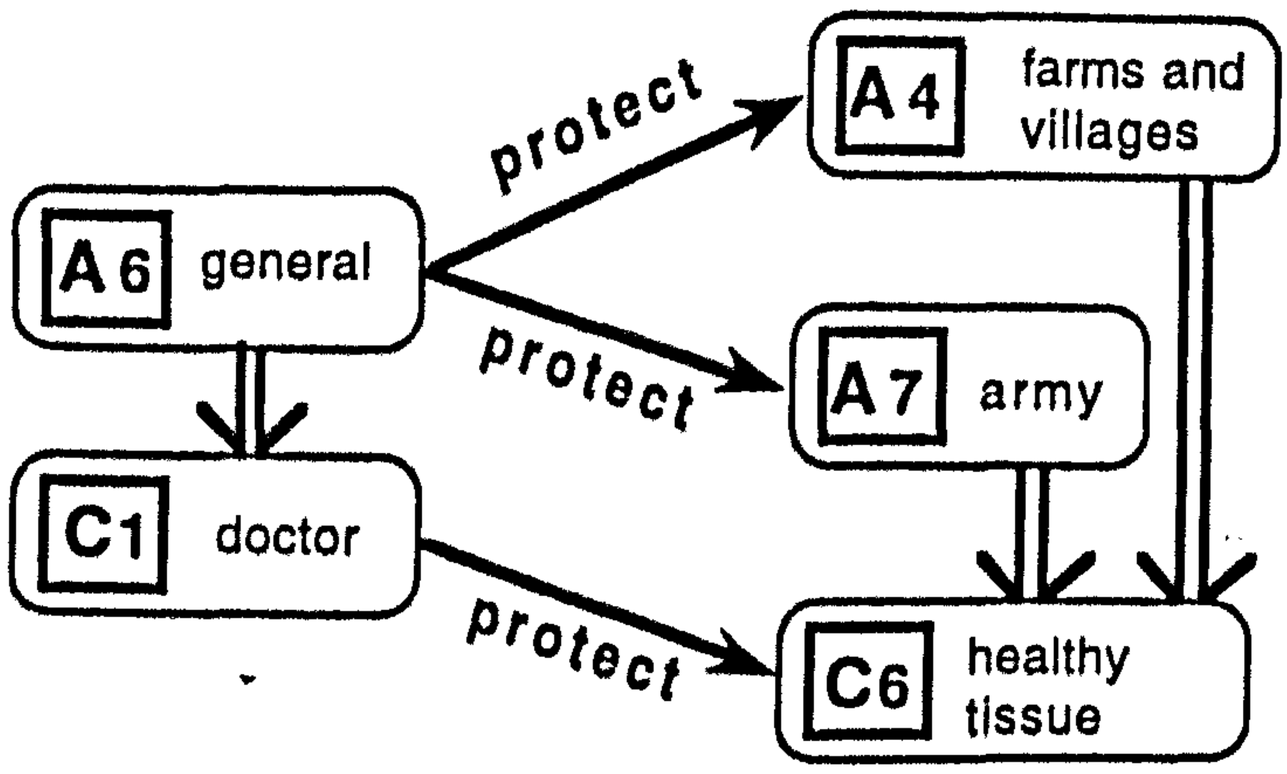


Figure 3.15. Mappings between 'healthy tissue' in the radiation problem and corresponding elements in the fortress problem which occupy a similar role (i.e. they are constrained by the higher-order predicate 'protect').

As part of the solution to the Fortress problem, the general divides his army into smaller units. How this operator can be applied to the Radiation problem, however, is not straightforward (figure 3.16); how can a ray be divided?

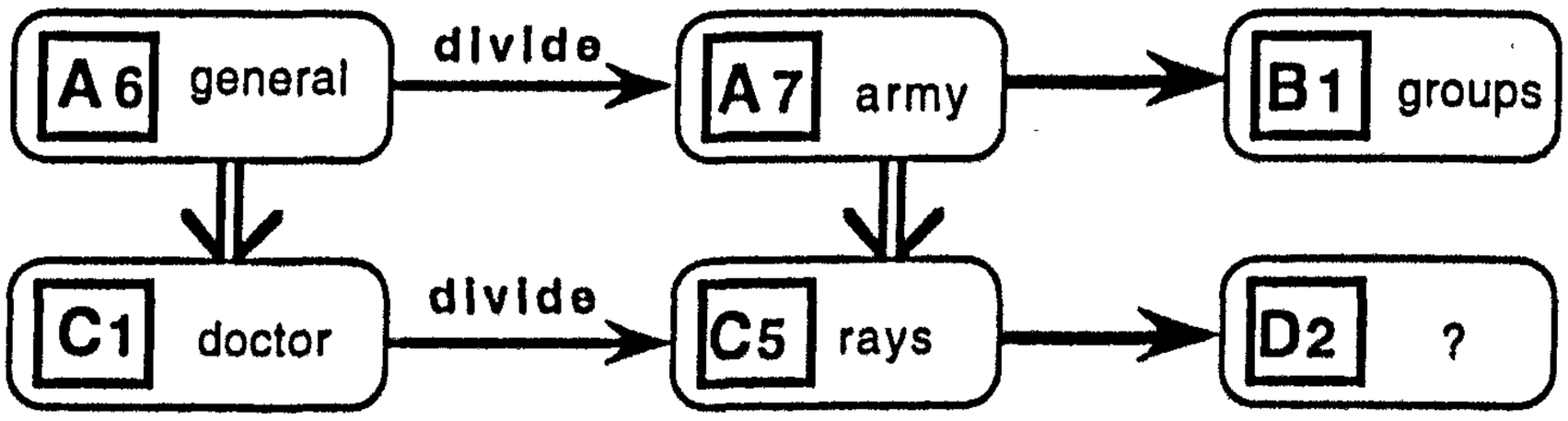


Figure 3.16. Mappings between 'general' and 'doctor', and 'army' and 'rays'.

The intensity of the rays could be reduced, but the solver is not told, and may not have the knowledge to infer, that this is a possibility. The doctor in the Radiation problem also has to find more ray-machines to bring the intensity back up to an adequate level. The general, on the other hand, does not call for more armies. The operator 'divide' has to be changed in the radiation problem to two operators 'reduce' and 'add' (figure 3.17).

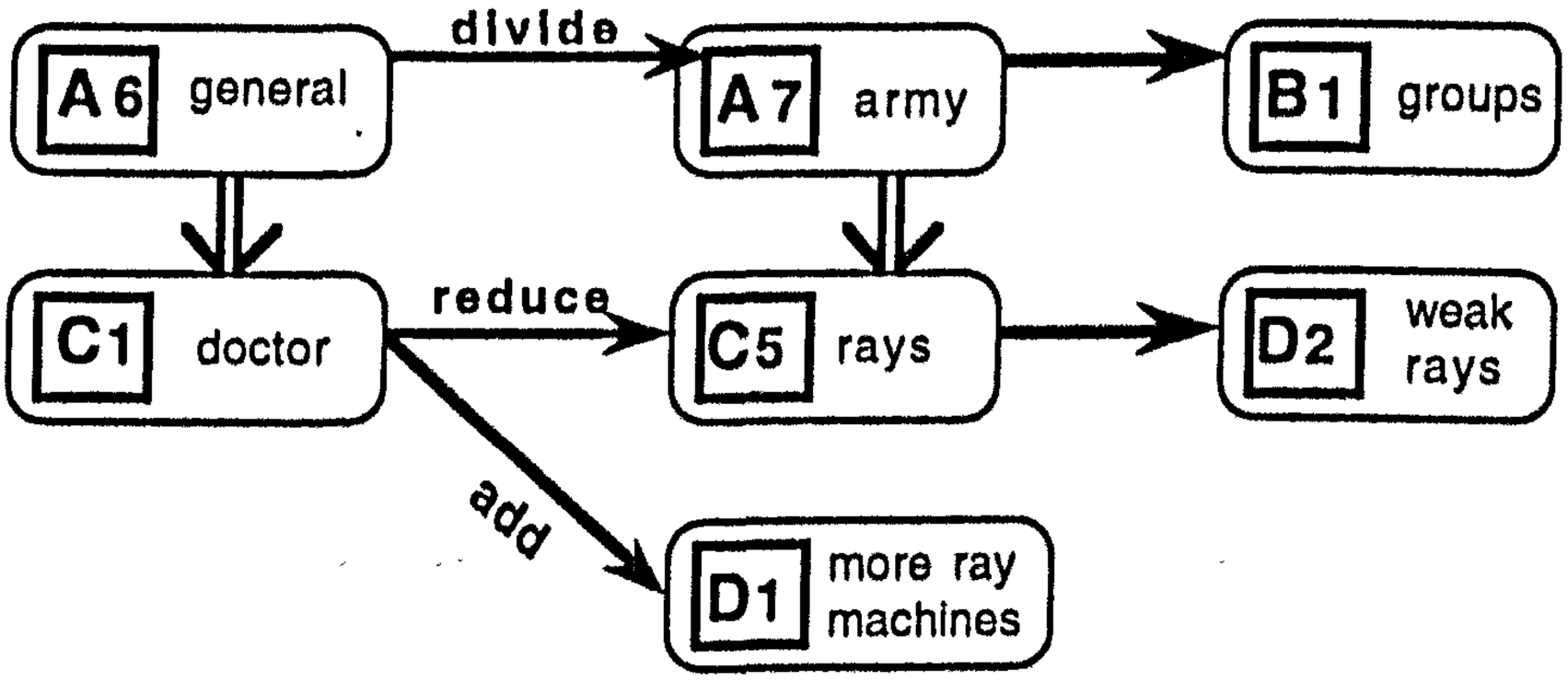


Figure 3.17. Adapting operators in the fortress problem to apply to radiation problem.

Both the groups and the machines have to be moved so groups should map to ray machines (the double arrow from B1 to D1 in figure 3.18, representing partial solutions to the example and exercise problems). Whereas the groups are moved to the heads of roads, there is no equivalent in the Radiation problem to 'roads'. Instead the ray machines are positioned around the patient (the operators in figure 3.18 are shown in ovals). In the Fortress problem the groups then converge simultaneously on the fortress, but it is the rays, not the machines in the Radiation problem, that converge on the tumour.

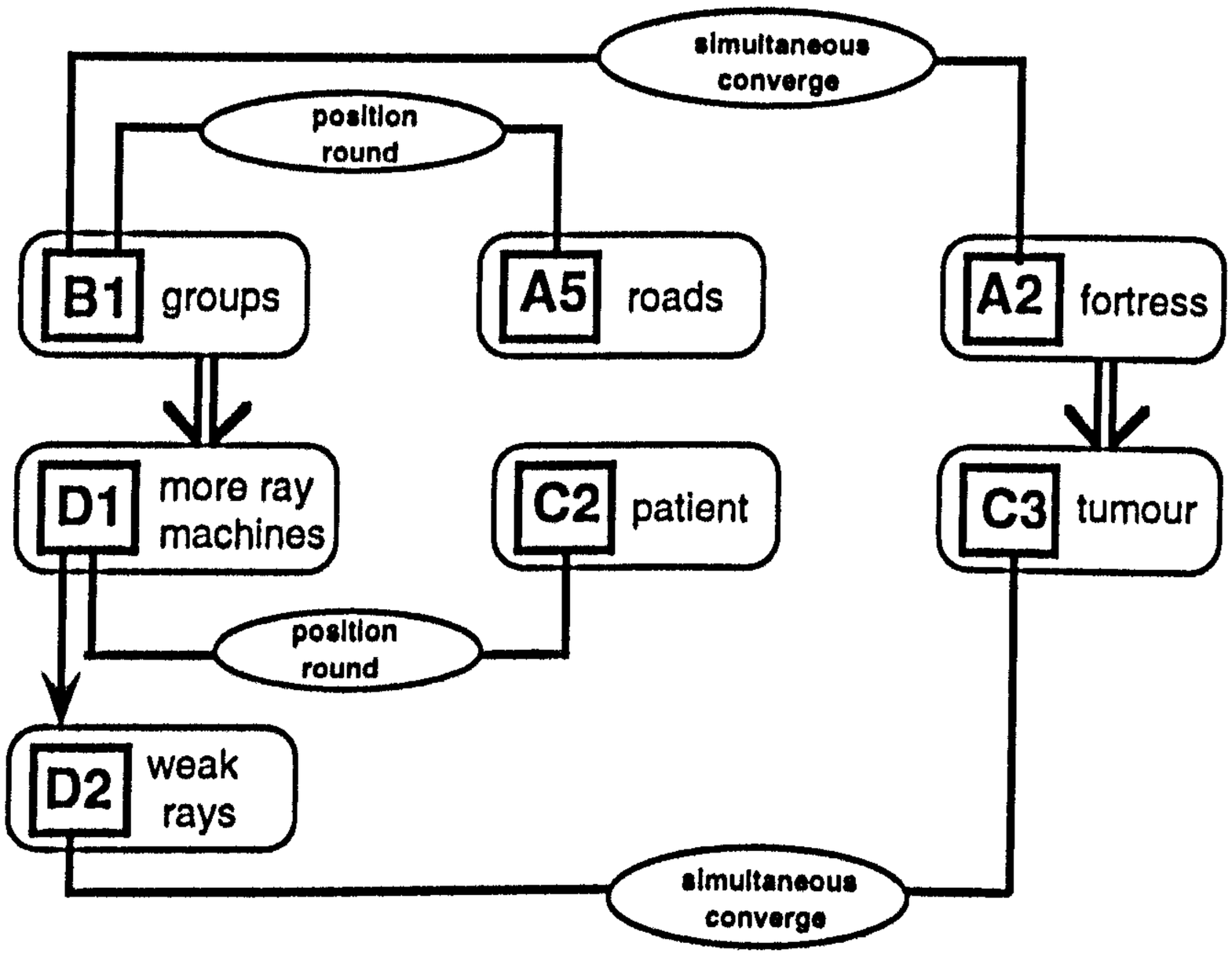


Figure 3.18 Blocked mappings between elements in the fortress and radiation problems.

The general has two aims: one is to capture the fortress and the other is to overthrow the dictator. The surgeon has to get at the tumour and destroy it. 'Capturing' and 'overthrowing' are not necessarily the same thing as 'destroying.' The solver has therefore to map and adapt aspects of the Fortress problem in order to solve the Radiation problem, but both 'capturing the fortress' and 'overthrowing the dictator' could map onto 'destroying the tumour'.

Adapting the Fortress problem to solve the Radiation problem is not simply a question of altering values and applying the same operators as were employed in the earlier problem. Further adaptations are required, and the solver has to infer what those adaptations are. It is mainly because of the requirement to manipulate and adapt the earlier problem that about 10% of Gick and Holyoak's subjects were unable to solve the Radiation problem even with a hint to use the Fortress problem.

4.2. Text analysis

Because of the way textbooks are designed, students are likely to have certain assumptions about how example problems are normally presented, that is, they are liable to expect that examples are presented as a problem statement followed by a solution, in that order. This schema is used to guide a student's search through the training material if they need to find information later. However, the presentation of worked examples in textbooks does not always follow such a canonical form. Instead, there are many variations, such as:

Problem Statement + Problem Solution
Problem Solution + Problem Statement
Problem Solution with no Problem Statement
Problem Statement with no Problem Solution

The variations in sequence increase when explanations or other forms of problem representation such as graphs, diagrams, tables, etc. are added to the text.

To illustrate elements of the interpretation theory as it can be applied to texts, I will be looking at examples from two LISP programming textbooks by, Hasemer & Domingue (1989) and Winston & Horn (1989), and an Artificial Intelligence course book by Eisenstadt (1978/1983) which formed part of a Cognitive Psychology course at the Open University. The latter was especially designed to be 'user-friendly' in the way the examples were presented.

The interpretation theory describes texts in terms of a number of concepts:

- example problem statement,
- example problem solution,
- explanations
- adequacy and comprehensiveness of explanations
- intermediate representations
- 'ghost' terms
- the problem givens
- blocks and impasses
- templates
- subgoals
- nested subgoals
- solution constraints
- inferences
- concepts
- operators

When used for protocol analysis, the theory also identifies:

- Experimenter interventions
- Experimenter generated examples
- 'understanding' processes

These are represented in the interpretation theory diagrams as shown in the summary figure 3.19 and described in detail thereafter.

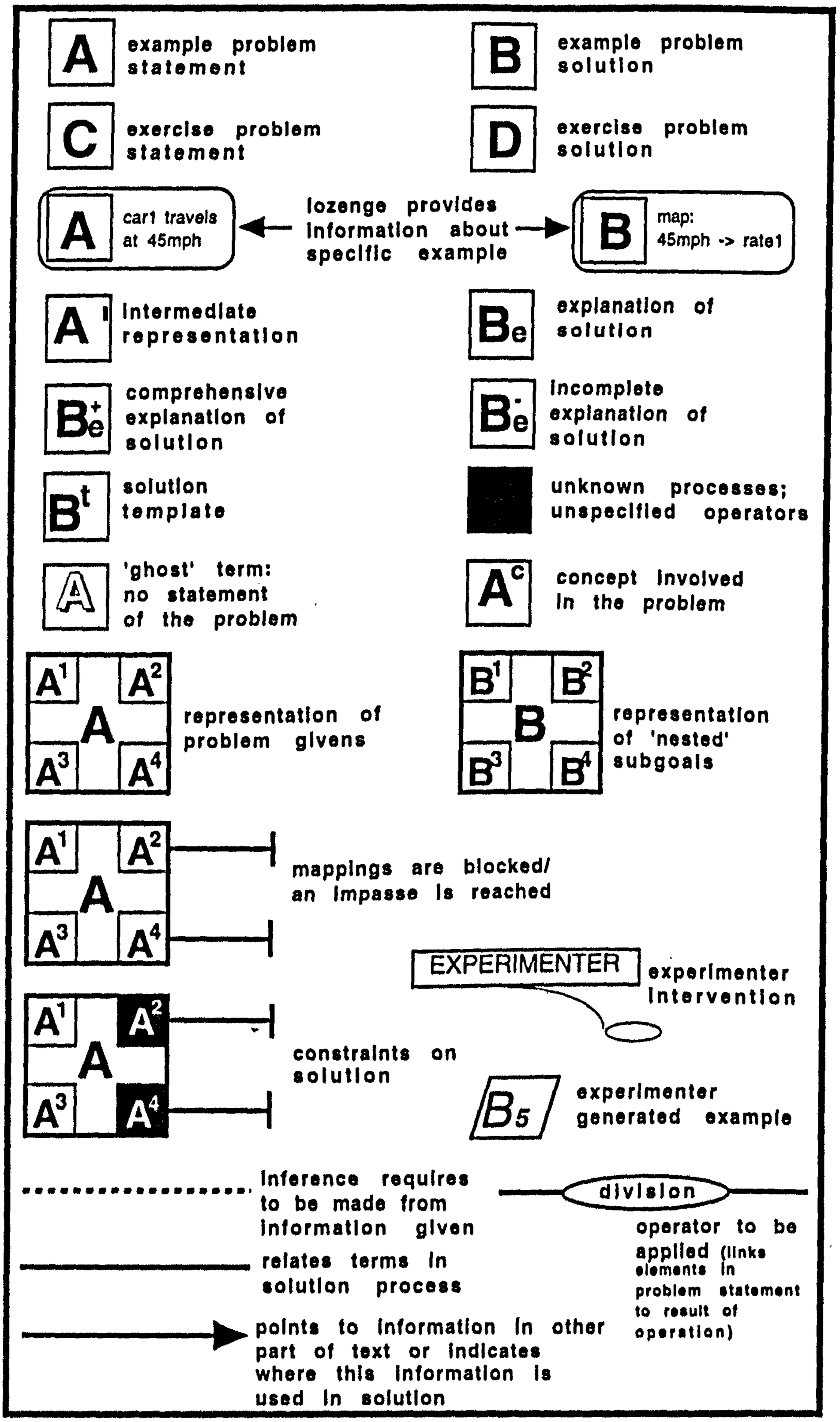


Figure 3.19. Summary of the notation used in the text and protocol analysis.

4.2.1. Problem statement and solution

The purpose of a science textbook is to provide the learner with a framework for understanding a particular domain. The text must attempt to show how the givens in the problem statement (the A term) can be translated into the correct form that the solution must take (the B term) by employing the rules or principles given in the rest of the training material. In figure 3.20 a link between the A and B terms is represented by a line from one term to the other. Although figure 3.20 provides an indication of a link, the nature of that link is not specified. (The types of link that can exist between terms are specified in more detail in later sections.)

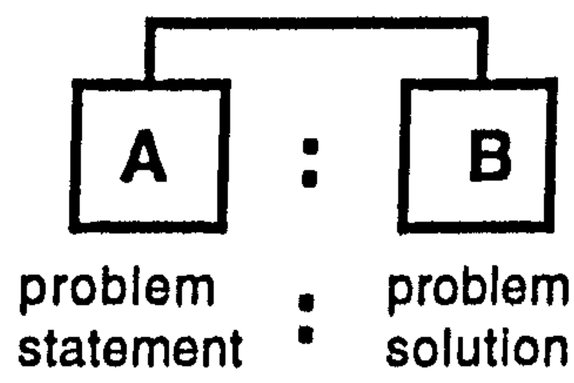


Figure 3.20. Linked problem statement and solution.

It is usual for the problem statement to precede the solution. However, where the reader is invited to look at a piece of code and the problem statement is given after it, the order is reversed. On page 64 of Hasemer and Domingue there is this example:

'A simple if expression might look like this:

```
(if (null q) (setf qq t)
   (setf qq nil))
```

This should be read as "If the list q is empty then set (setf qq t), else (setf qq nil)."'

In such a case the B term (the solution) is given in the text before the A term (the statement). Notice also that in this case there is no explicit link between the solution and the problem statement. It does not, for example, explain the link between (null q) and 'the list q is empty'; that the order of terms represents 'if... then... else...'; what (setf qq t) means; and so on.

This state of affairs is represented in figure 3.21.

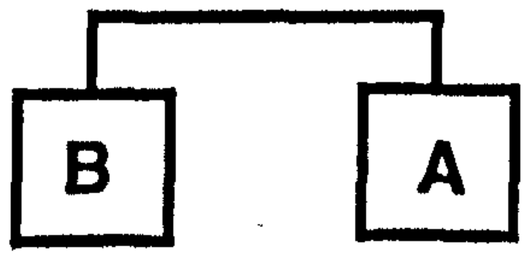


Figure 3.21. Representation of a solution followed by problem statement in a textbook with no explicit explanation of the relation between them.

The reader is therefore left to infer relations between the problem givens and the form of the code. The nature of inferences is an important consideration in the design of instructional material. Things have to be spelled out in more detail at the beginning when the reader is presumed to know very little about the new domain. The extent to which things have to be spelled out is an empirical question, as is the question of how much any spelling-out can be relaxed in later sections.

The example problem represented in figure 3.20 violates the reader's schema for how a text is organized: if, when solving a later exercise problem, readers access that example solution, they would expect to find a statement of the problem *before* the solution and consequently would go backwards in the text to find out what the problem is. Unless the text is well signposted, they may find a problem statement for a different problem and try to relate that to the current solution and thus become even more confused.

4.2.2. Explanations

Textbooks explain things in various ways. They can, for example, explain how a particular type of problem should be tackled. This constitutes an explanation of the problem from a general point of view. They can also explain how the solution is generated from the problem givens: this is an explanation of the solution and is much more specific. Explanations are indicated by the letter 'e' in the term box. Where the problem is explained the 'e' subscript appears in the A term box thus:  and where the solution is explained it appears in the B term box:  .

The following example comes from Eisenstadt (1978):

A: Here is an example of a procedure called BOOZETEST, which finds out whether a person **Be:** (actually the node to which BOOZETEST is applied) **A:** drinks a beverage which contains alcohol.	EXAMPLE PROBLEM STATEMENT + EXPLANATION OF SOLUTION

Be: To do this it first uses pattern-matching to find out what beverage (if any) the person consumes. Then if it finds such a beverage, it tries to find out whether that beverage contains alcohol. Notice particularly how the variable * is used in step 2 to keep track of the node retrieved as a result of the pattern-matching in step 1.

EXPLANATION OF SOLUTION

B: SOLO: TO BOOZETEST /X/
...: 1 CHECK /X/ --- DRINKS ---> ?
.....1A IF PRESENT: CONTINUE
.....1B IF ABSENT: PRINT "NO INFORMATION"; EXIT
...: 2 CHECK * --- CONTAINS ---> ALCOHOL
.....2A IF PRESENT: CONTINUE
.....2B IF ABSENT: PRINT "NON-ALCOHOLIC"; EXIT
...: 3 PRINT "AHA, " /X/ "IS A REAL BOOZER"
...: DONE

SOLUTION

Be: The control-statements in BOOZETEST are used in a manner analogous to the way they were used in WEAKASSESS and STRICTASSESS, presented in section 6.4. Substeps 1A and 2A specify CONTINUE, because that is the main pathway through a successful sequence of CHECKs, down to the message which gets printed out at step 3.

EXPLANATION OF SOLUTION

The text above can be represented as in figure 3.22 where A represents the problem, B the solution, and Be represents the explanation of the solution. The link between the A term and B term represents the fact that the relation between the problem givens and the solution is explained. The links between the A and B terms and the Be term represent the fact that aspects of how SOLO evaluates the code are explained both in the problem statement as well as after the solution. The arrow to STRICTASSESS shows that the text is directing the reader to another part of the text.

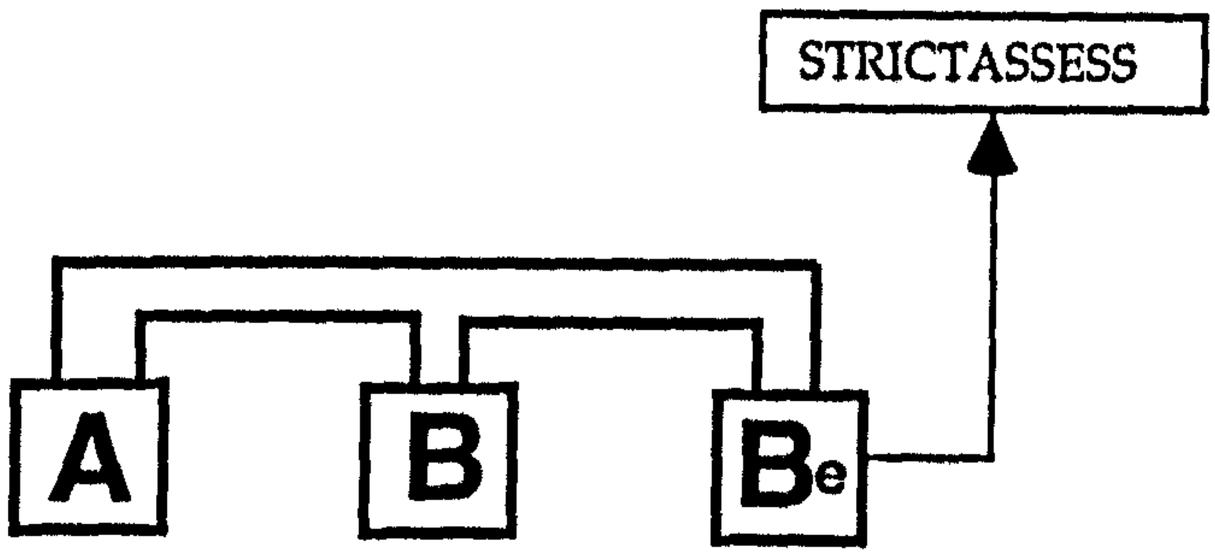


Figure 3.22. Representation of explanation of 'BOOZETEST' in Eisenstadt (1983).

4.2.3. The adequacy versus comprehensiveness of explanations

Writers of textbooks have to make their explanations complete or comprehensive enough for their readers to understand. The early chapters of a textbook in a new domain have to be particularly comprehensive in their treatment of novel concepts since few assumptions can be made about a reader's prior knowledge. In later chapters the explanation can be less comprehensive but must be adequate for the reader to understand them, which means that the writers must make some assumptions about what has been understood from previous chapters. The balance between a comprehensive (and possibly tedious) explanation and an adequate (and possibly dense) explanation is a fine one. Part of the difficulty facing writers of textbooks is that they are ipso facto experts, and so have a great deal of compiled or schematic knowledge about their domain of expertise (Adelson, 1981; Anderson, 1983; Ehrlich and Soloway, 1984; Novick, 1988; Owen and Sweller, 1989). Experts have to reconstruct the declarative knowledge from which their procedural knowledge was built in order for novices to understand all the steps in a particular procedure.

In the example from Eisenstadt (1978) in section 4.2.2, part of which is reproduced here, there is an assumption that the reader understands what a 'control-statement' is and how it was used in the previous examples (WEAKASSESS and STRICTASSESS). However, if the readers have any doubt about what a control-statement is, they are referred back to the section where these examples were introduced:

The control-statements in BOOZETEST are used in a manner analogous to the way they were used in WEAKASSESS and STRICTASSESS, presented in section 6.4.	EXPLANATION OF SOLUTION

Where an explanation is provided which is incomplete in that it makes assumptions about the knowledge of the learner, it is represented in our theory with a '-' above the 'e' subscript in the B term. Where an explanation is complete, that is where the reader does

not have to make any inferences, it is represented by a '+' above the 'e'. These are illustrated in figure 3.23.

Figure 3.23. Complete and incomplete explanations

4.2.4. Intermediate representations

Learners can be helped to understand a new concept when they are given another way of representing the problem and its solution. For instance, an example problem statement may be followed by an intermediate representation of the problem. Thus, in Duncker's Radiation Problem , if a solver were asked to imagine the spokes of a wheel as an aid in solution, this would be counted as an intermediate problem representation. This representation is presumed to enable the solver to abstract out important features of the problem which can thereafter be used to solve a number of Gick and Holyoak's 'convergence' problems.

Where the writer is trying to help the reader understand the nature of a problem by some form of intermediate representation this is indicated by a prime symbol, as in A'. Where it is the explanation of the solution that is being facilitated, it is represented by B' in figure 3.24.

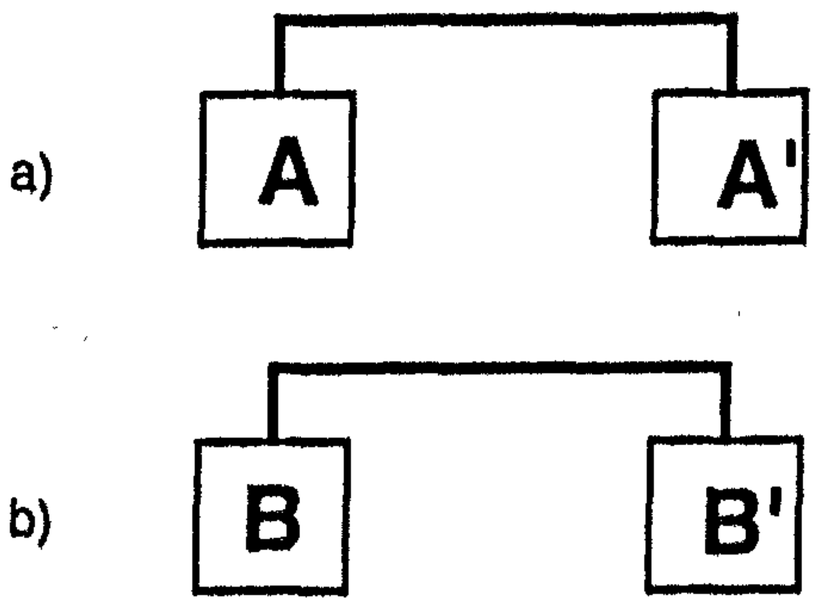


Figure 3.24 Problem statement (a) and solution (b) followed by an intermediate representation.

Eisenstadt (1983a) provides a series of diagrams to illustrate flow of control in SOLO as part of the explanation of how the following 'INFECT' procedure works:

```
SOLO: TO INFECT/X/
...:1 NOTE/X/ --- HAS ---> FLU
...:2 CHECK/X/ --- KISSES ---> ?
.....2A IF PRESENT: INFECT *; EXIT
```

.....2B IF ABSENT: EXIT
...: DONE

Figure 3.25 shows one of the diagrams used:

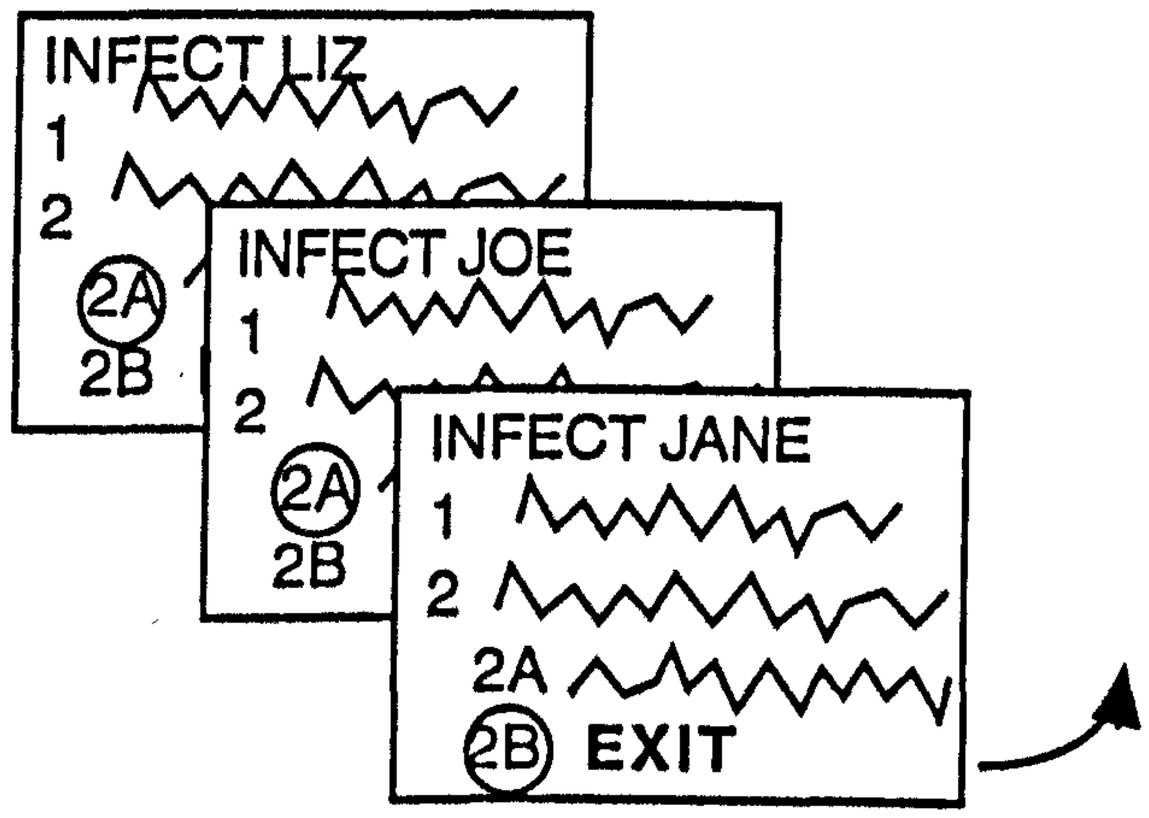


Figure 3.25. A 'snapshot' of the operation of the INFECT procedure (Eisenstadt, 1983a).

An explanation of the problem (Ae) followed by an explanation of the solution (Be) which then refers the reader to the intermediate representation (B'), can be shown in the framework as in figure 3.26:

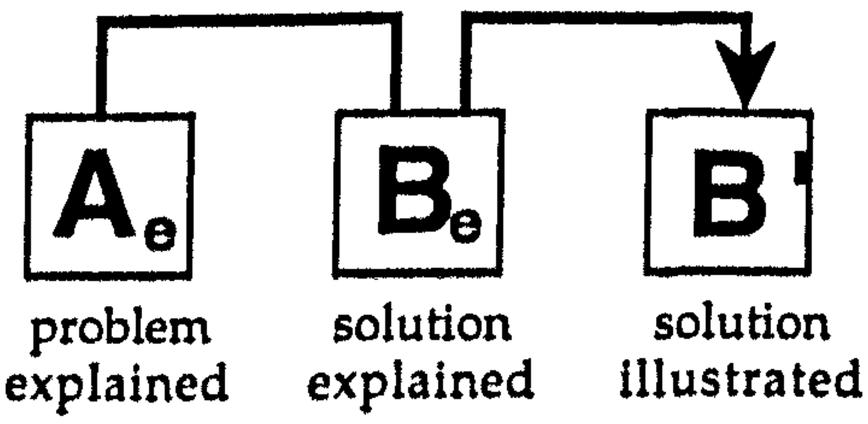


Figure 3.26. Representation of the previous explanations and intermediate representations in Eisenstadt (1983a).

4.2.5. 'Ghost' terms

Sometimes a solution is presented with no explicit statement of the problem itself. Section 4.2 of Hasemer and Domingue present 8 examples only half of which have statements of what the problem is. Another example can be found on page 106, which says, 'Look carefully at this function.' This is followed by a piece of LISP code and an explanation of how it works. The reader is left to infer what it is supposed to do. Very occasionally the problem solution is not made explicit: 'The answer to this question is found in the same way as the previous one.' Wherever the reader is left to infer or imagine what the

missing term should be, this is referred to as a 'ghost' term and is presented in a shadow format (figure 3.27). Both Lisp textbooks often miss out statements of the problems presented. Textbook writers have to be very sure that readers can readily infer what the problem is. First, if they later access this solution while trying to solve an exercise problem, they might expect to find a statement of the problem before the solution and search backwards in the text to find out what the problem is. Second, the reader is prevented from making any mappings since there is nothing to map.

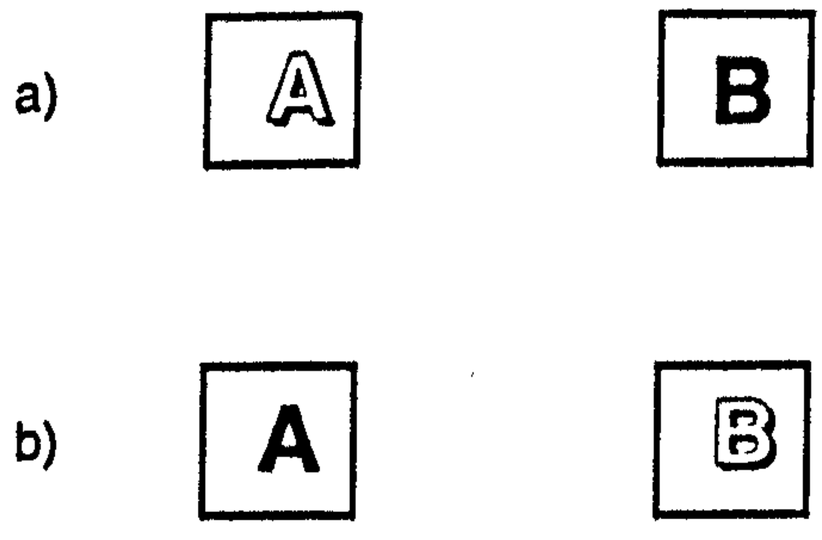


Figure 3.27. Representing 'ghost' terms.

4.2.6. The problem givens

Problem statements are made up of a number of problem givens. These are the features of a problem which are necessary to the solution. An algebra problem involving speeds, times and distances may include such information as: *car1 travels at 30 mph, car2 travels at 40 mph, car1 leaves at 10.00 a.m.* and *car2 leaves at 11.30 a.m.* Each of these givens is substituted for the variables of an equation which, when solved, will produce the answer to the problem. In the interpretation theory, problem givens are shown as being 'nested' within the A term. In figure 3.28 A_1 represents the speed of car1, A_2 that of car2, and A_3 and A_4 are the departure times.

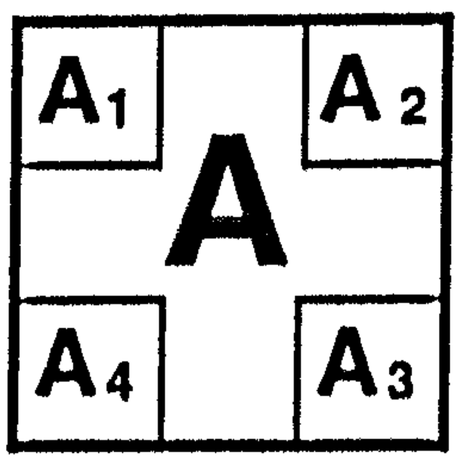


Figure 3.28. 'Nested' problem givens.

4.2.7. Blocks and impasses

Situations can arise in a text where the givens in a source problem cannot be mapped to the target, perhaps because they have been stated in a different form. Alternatively, the

solution to an earlier problem is not relevant to the current one and yet the solver is attempting to use it. The framework indicates these mappings as being 'blocked' (figure 3.29; see also subsection 4.2.11).

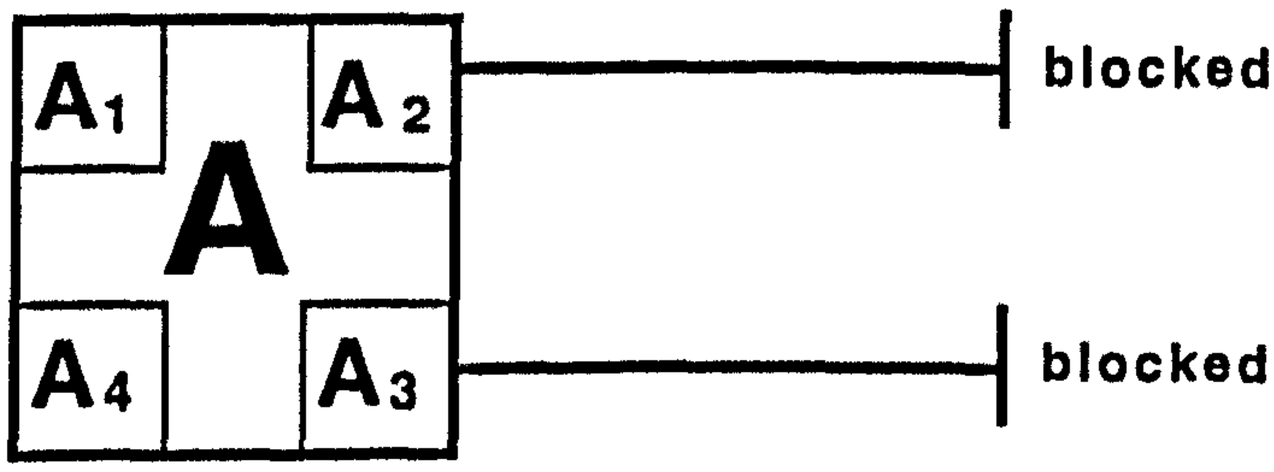


Figure 3.29. Blocked mappings.

The same representation is used when analysing verbal protocols. There are times when a solver reaches an impasse and does not know what to do or where to look for help. In such cases also the blocked line is used to represent an impasse.

4.2.8. Templates

Example problems are sometimes given in the form of a 'template'. Such partially instantiated solutions are designed to show what the layout of particular programming structures should look like. In the interpretation theory, templates refer to any structure which demonstrates the syntactic form a solution should take. An example of a template can be found on page 66 of Hasemer and Domingue where the following expression occurs:

```
(cond     ((null lis) (<do something>))<action>...<action>)
          (t (<do something>)))
```

In our framework these templates are represented with a *t* superscript as in figure 3.30.

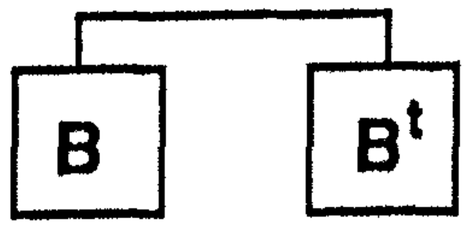


Figure 3.30. A solution followed by a solution template.

4.2.9. Subgoals

In many domains such as mathematics, physics and programming, solving a problem means attaining a number of subgoals on the way to the overall goal. In complex problems where several subgoals (which may in turn contain embedded subgoals) have to be attained, all subgoals are represented as being 'nested' within the B term (or the D term if the exercise

problem is being analysed). On page 66 of Hasemer and Domingue there is a worked example which prints the value of the variable *item*, then sets the value of *qq* to the *cdr* of *item* (that is, the value of *item* minus its first term) and prints that. Since this is the first example solution in this section it is numbered B1.

((print item) (setf qq (cdr item)) (print qq))

There are three subgoals to be achieved each of which is identified by a letter:

$B1^a$: (print item)

$B1^b$: (setf qq (cdr item))

$B1^c$: (print qq)

$B1^b$ also contains an embedded subgoal (*cdr item*) which we can label $B1^{b1}$. We can now represent subgoal $B1^b$ as in figure 3.31.

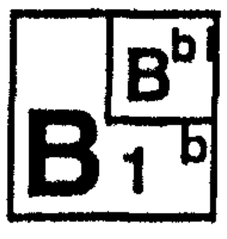


Figure 3.31. One subgoal represented as nested within another.

4.2.10. Nested subgoals

On page 67 of Hasemer and Domingue there is an explanation of how to write a function definition which leads up to a piece of LISP code defining a function (named *foo*) which, when called, will do exactly what the earlier program, represented by B1 in section 4.2.9, did. Since this is the second example provided this can be represented by B2 but, in order to indicate that the second example contains the first one nested within it, it will be shown as in figure 3.32a.

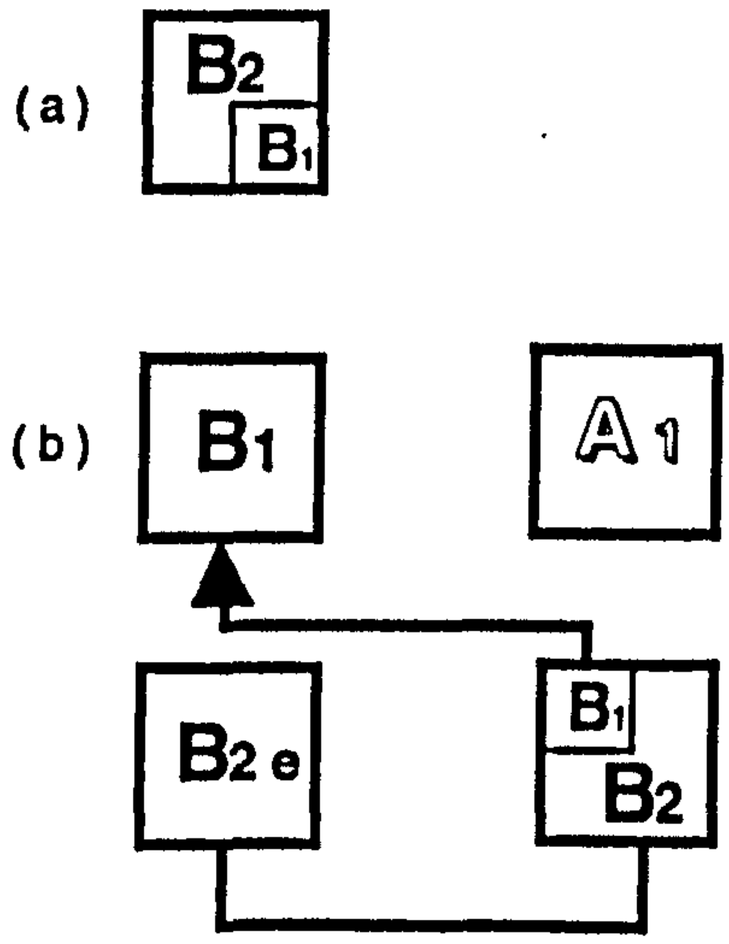


Figure 3.32. Representation of a section of text in which a previously mentioned piece of code is incorporated in a new piece of code as a subgoal.

In the text on page 66 of Hasemer and Domingue, example B_1 is presented with no indication of the problem statement. The example is followed by an explanation of a function definition, shown as B_{2e}, followed by the specific example solution, B_2, which includes B_1. In figure 3.32b we can represent the path taken by the text in the section dealing with function definitions. The writers assume that the readers have understood the LISP code represented by B_1 (the earlier example with its own nested subgoals).

The final function definition on page 67 is shown in figure 3.33.

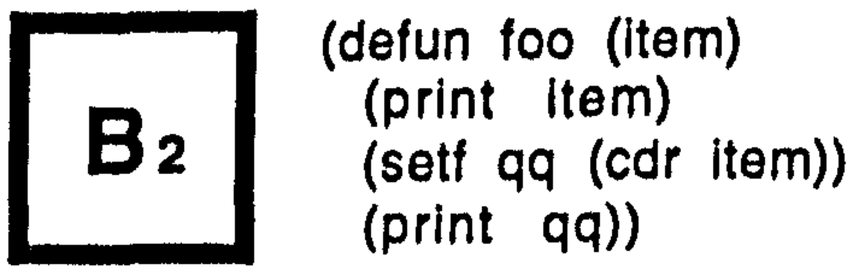


Figure 3.33. The function definition *foo* on page 67 of Hasemer & Domingue (1989)

Writing a function definition involves writing the *macro* defun followed by a function name (foo) and a parameter list (item). The next three lines give the three actions which the function carries out. Figure 3.34 shows that the first subgoal is to write the function name and parameter list and the second is to specify what the function does.

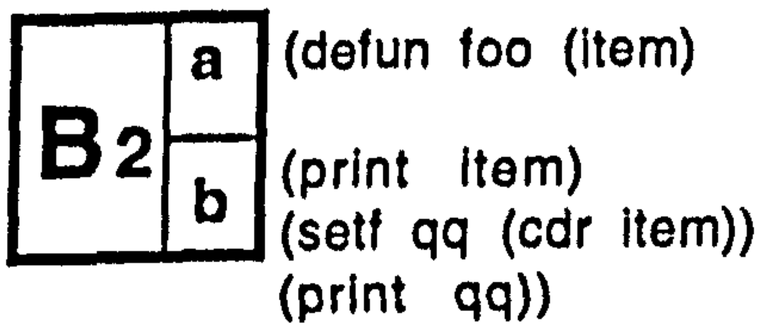


Figure 3.34. Main subgoals in the *foo* definition.

$B2^b$ also contains a number of subgoals labelled 'i', 'ii', and 'iii' in figure 3.35. $B2^{bii}$ contains a further subgoal, that of generating the cdr of item. Figure 3.35 demonstrates how subgoals at different levels can be visualized in this example solution.

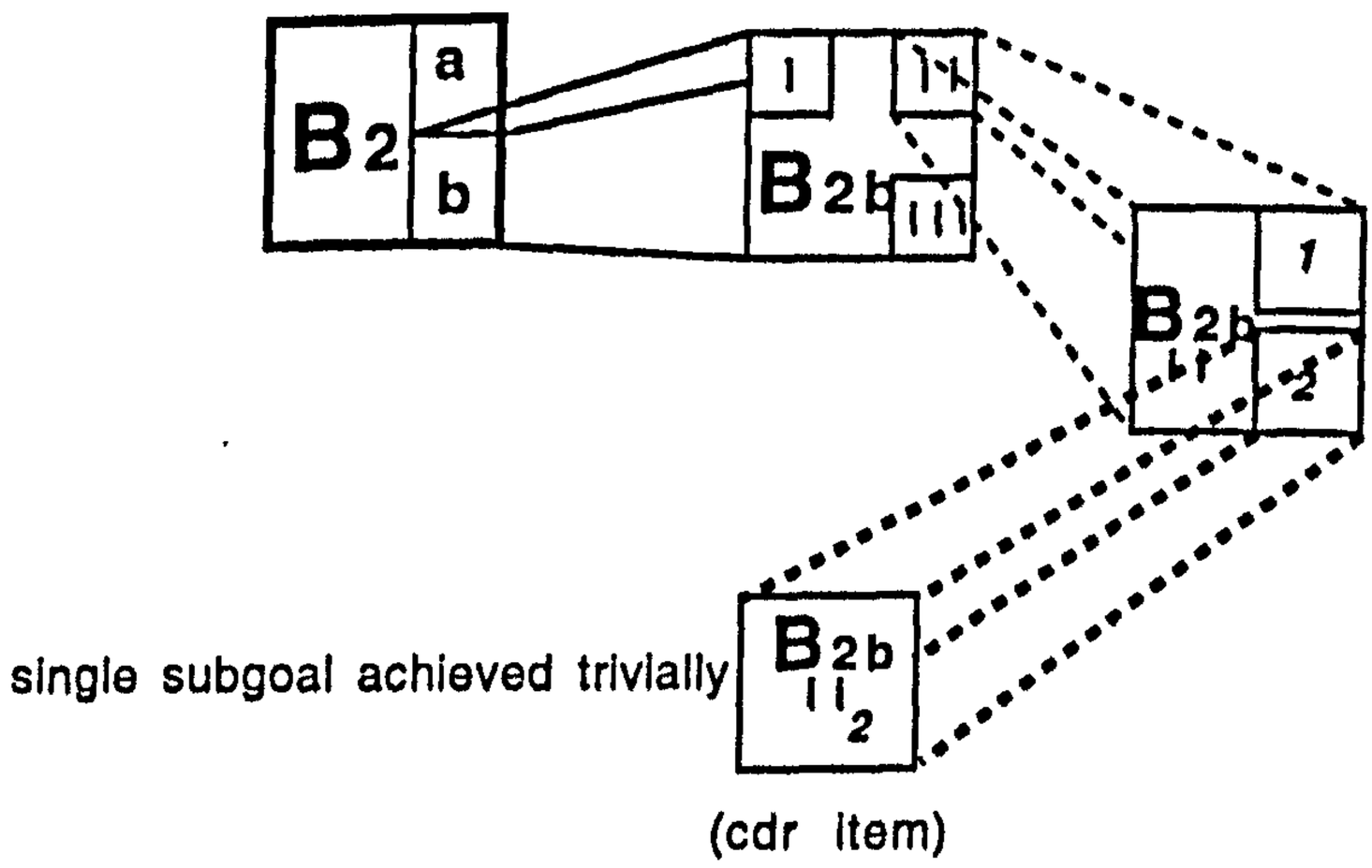


Figure 3.35. Nested subgoals.

4.2.11. Constraints

Some problems include explicit constraints or restrictions on a solution path. Versions of the Radiation problem include the constraint that a ray may not pass down the oesophagus and that surgery may not be used to open up the stomach to get at the tumour. Constraints are represented as a 'photographic negative' with the black and white fore- and background reversed. Since constraints prevent a possible solution, the paths from the constraints are shown as being blocked (see subsection 4.7). In figure 3.36 two constraints for the Radiation problem are represented. Most problems have implicit constraints, and it is part of the problem solving process to find out what these constraints are. In the problem solving episode in Chapter 1, for example, the subject was unsure whether it was necessary to represent the fact that there was a water tank in the attic. As far as she was concerned the problem was underconstrained, and part of her problem solving involved discovering what the constraints were. Only when constraints are *explicitly* stated are they represented as in figure 3.36.

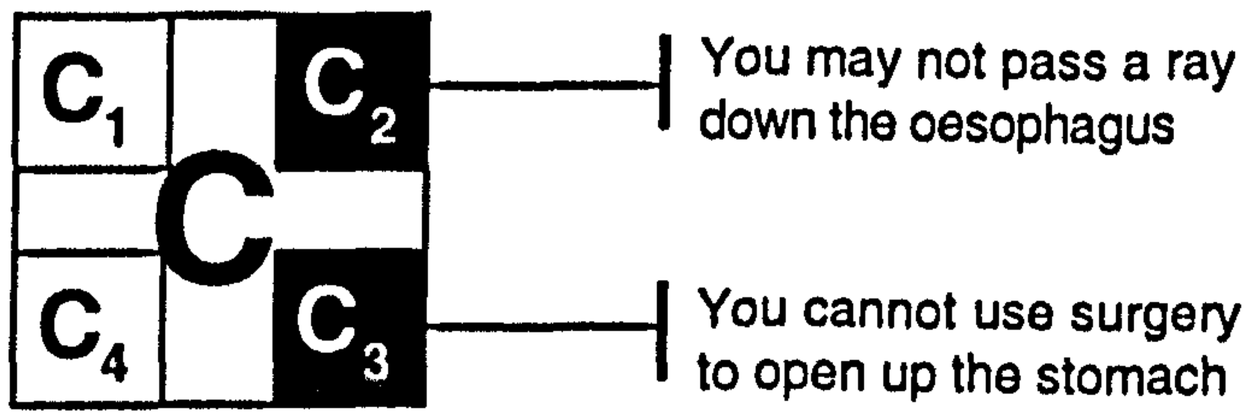


Figure 3.36. Explicit solution constraints.

4.2.12. Inferences

In later chapters of textbooks writers assume a certain amount of knowledge on the part of the student. This means that the student is often required to make inferences when presented with example problems. An example of this is the *Distance = Rate x Time* problem presented by Reed, Dempster & Ettinger (1985) in which two vehicles leave the same place at different times (10.00 a.m. and 11.30 p.m.) and travel at different speeds. The second vehicle eventually overtakes the first. Since both vehicles travel the same distance, the appropriate equation to use is $Rate_{vehicle1} \times Time_{vehicle1} = Rate_{vehicle2} \times Time_{vehicle2}$.

In figure 3.37, A^1 and A^2 represent 11.30 and 1.30 respectively. In the equation which underlies the solution, the time taken by vehicle 1 is shown in the explanation as $(t + {}^3/_2)$ There is no indication of where the '$^3/_2$' comes from. The solver is left to make the inference that the times mentioned in A^1 and A^2 somehow relate to the '$^3/_2$' given in the solution. When the solver has to infer which problem givens are relevant to the solution, this is represented by the dotted lines shown in the figure. The number '$^3/_2$' is arrived at by applying the subtraction operator to the givens, and then converting that to a fraction. Thus, there are two operations that are not explained in the example. The 'black boxes' in figure 3.37 are used to show that there is no indication of what the relevant operators are .

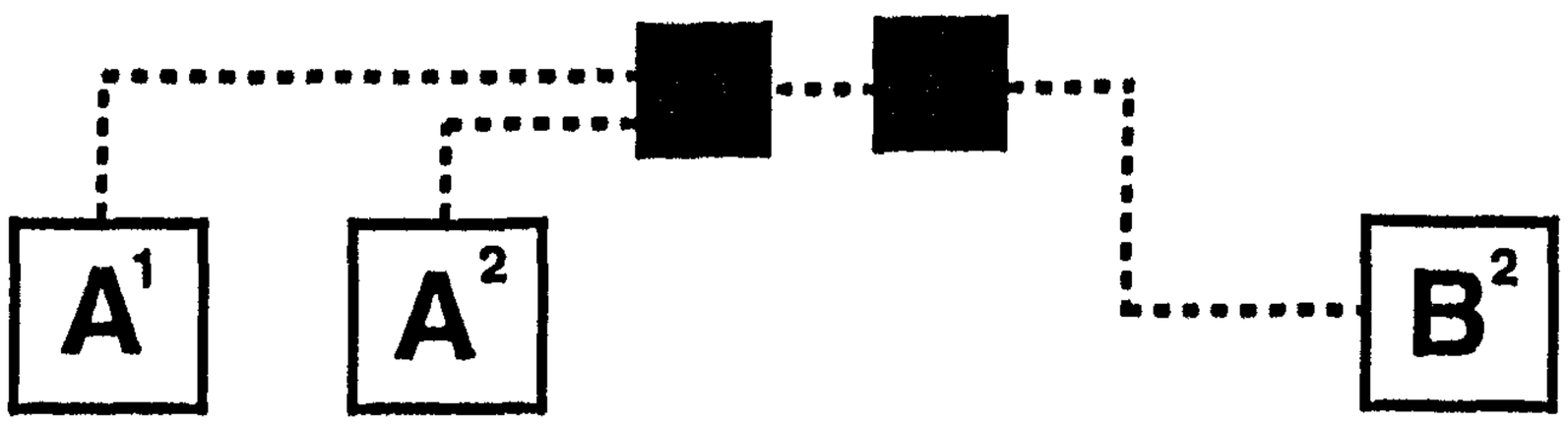


Figure 3.37. Making inferences from problem givens.

4.2.13. Concepts

Worked examples provide illustrations of how concepts can be manipulated to provide a solution. Examples of this are the use of recursion in LISP functions or how the concepts of *Distance* and *Time* can be manipulated to find *Speed*. It is, therefore, sometimes useful to analyse problem statements in terms of the concepts which they embody. By doing so we can represent how the problem relates to the concepts which it is attempting to illustrate. In linear algebra word problems discussed in the next section, the distance a vehicle travels is related to the rate at which it travels and the time it takes. If the time a vehicle leaves is mentioned first in the problem statement, this can be represented as A^1 in figure 3.38. If its rate is then given, this would be represented as A^2. The 'c' indicates that these problem givens are concepts (the specific concepts are shown in the rectangles).

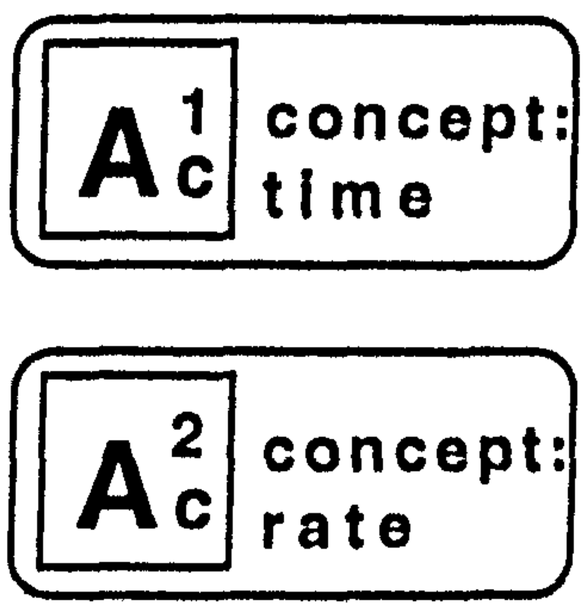


Figure 3.38. Representing concepts.

4.2.14. Operators

The initial state of a problem can be transformed into a new state by applying operators to the problem's givens. The new state may be the goal state (the solution) or a subgoal along the solution path. For example, applying a multiplication operator to the figures for *Time* and *Speed* in a *Distance-Speed-Time* problem would yield distance travelled. In the interpretation theory, the operator is represented as an oval superimposed on the link between problem givens and solution. The specific operator applied is written within the oval. In figure 3.39, if the multiplication operator is applied to the two concepts Ac^1 and Ac^2, this would yield the distance (Bc^1). Similarly if the concepts were replaced by specific figures such as 30mph for rate and 3 hours for time then the result would be 90 miles.

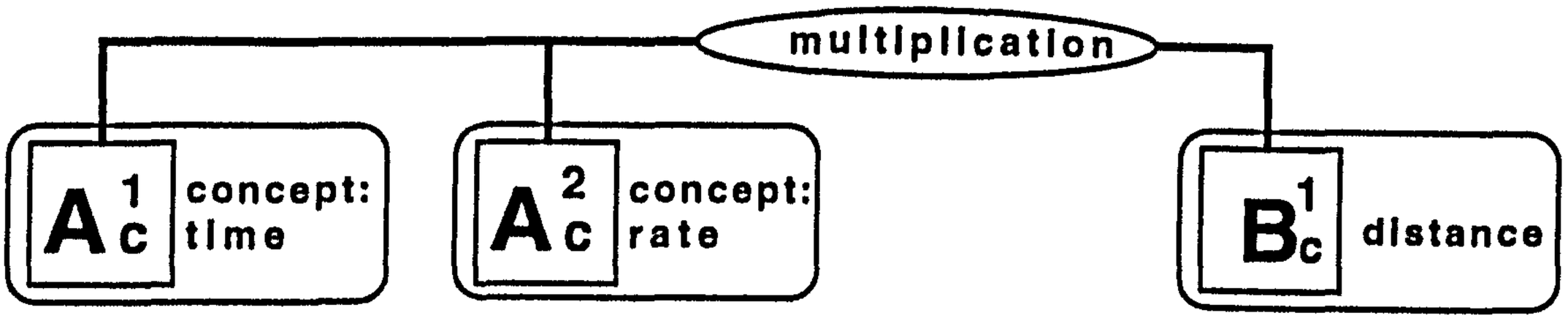


Figure 3.39. Applying operators to a problem givens.

Up until now the discussion has been mainly about text analysis. The same framework can be used in analysing students' problem solving behaviour. The remaining codings apply specifically to the analysis of verbal protocols.

4.2.17. Understanding processes

In Chapter 1, the subject found a candidate example problem which she used in order to understand how recursion was instantiated in an example. Part of the verbal protocol is reproduced below.

a)

> I'm looking on page 80, em,
> which has given a new definition of the **understanding training problem solution**
> INFECT procedure
> so that it keeps on recurring through it.

Here she shows evidence of simulating the behaviour of the example solution. At other times she simulates the behaviour of her own (partial) solutions.

On other occasions she spends some time trying to understand the problem statement. That is, she tries to understand what constraints on solutions apply.

b)

> Well, my problem is that, at the moment, the database doesn't contain the fact that there is a water tank in the attic, and therefore it's difficult to have... I mean, I was thinking, you see, of starting something like 'CHECK WATERTANK IN ATTIC BURSTS.' Well, I mean, obviously it wouldn't say that, but, I mean, something that would check that sort of thing. But since the database at the moment doesn't actually have the fact that there is a water tank in the attic, then, em... that makes it a wee bit difficult, doesn't it?

During problem solving episodes students may read through a problem statement or solution, or attempt to simulate the behaviour of a solution program, as in a) and b) above. Such episodes are classified as 'Problem Understanding' and in protocol analysis they are represented as in figure 3.40a and 3.40b. The circular arrow represents the fact that the subject has found an example problem solution (B2) and is re-reading it or simulating its behaviour if it is a computer program. The figure in the circle indicates that this is the 18th problem solving 'episode'. Similarly, in figure 3.40b, the subject is trying to understand the exercise problem statement (C), and this is the 4th such problem solving episode.

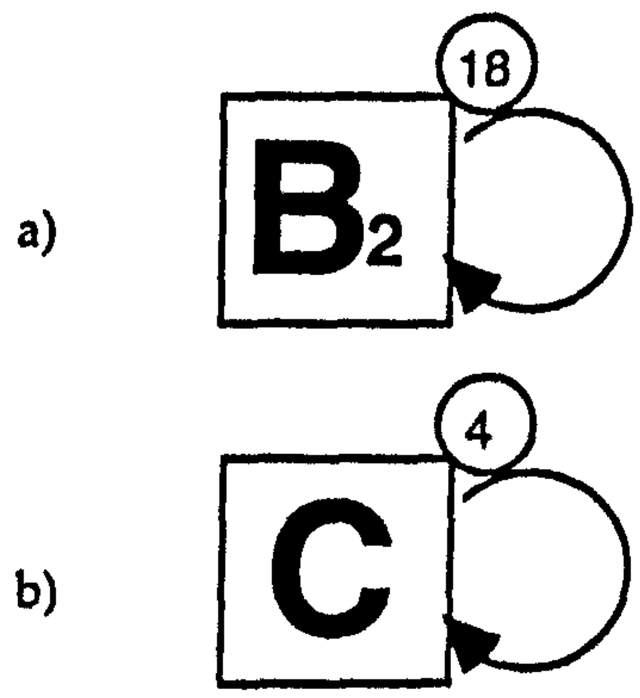


Figure 3.40. A problem solving episode in which the subject rereads or tries to understand an earlier problem solution. (a); or re-reads the problem statement. (b)

4.2.15. Experimenter Interventions - The Lifebelt

In experiments, an experimenter might intervene to *help* the subject rather than just to prompt the solver to reveal what he or she is thinking. In such cases the experimenter is actually teaching the subject. The student in a sense is being thrown a lifeline rather than being left to find the answer from the text or from memory. The following example is taken from a protocol presented in Anderson, Farrell & Sauers (1984). In this episode, the subject (B) is having difficulty understanding an example solution. To help her, the experimenter (E) reminds her of an earlier example she had seen in a previous chapter of the Winston & Horn (1981) textbook, and is asking her to compare the two problems.

E: Just like with an exercise before, remember we had to get PEAR out of [?] Get B out of LIST1. Yes, it's equivalent to that one.
B: Ok, so... at this point , LIST1 has the value of B R.

Experimenter interventions of this kind are represented by the lifebelt illustrated in figure 3.41.

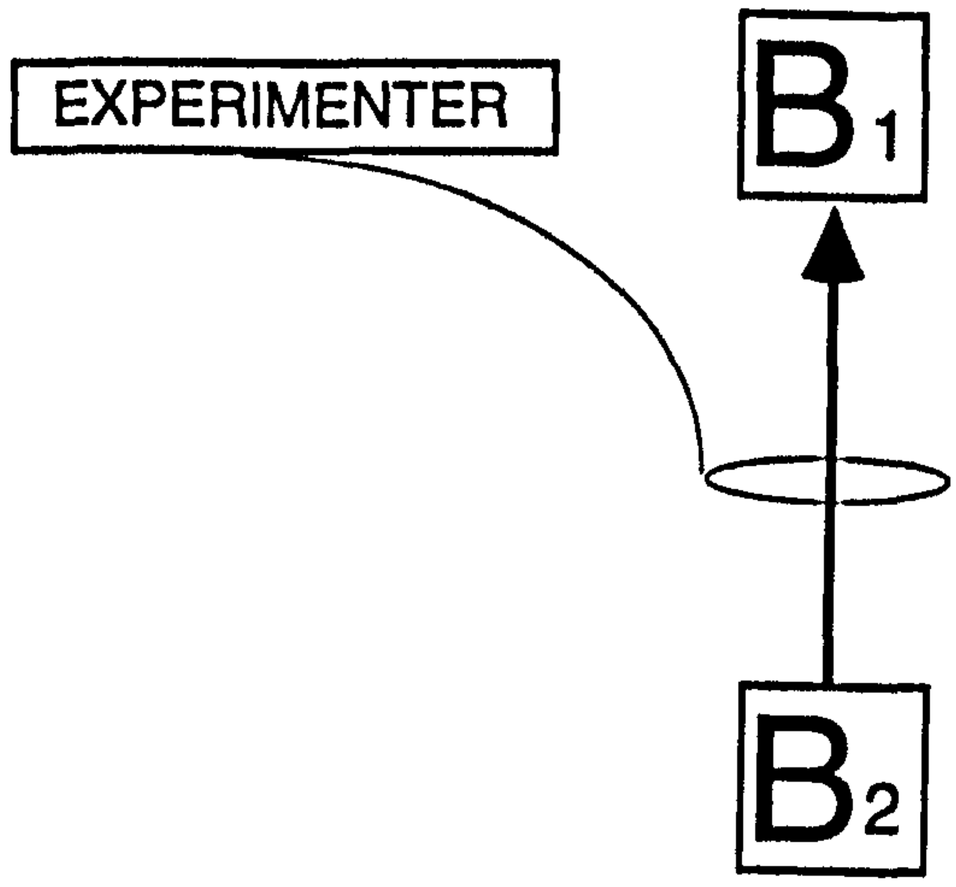


Figure 3.41. The 'Lifebelt'. The experimenter leads the subject to make some kind of mapping between examples.

In this example the experimenter intervenes to illustrate some concept instantiated in example B2 which is also in B1. The subject is being led to map the two solutions or parts of the two solutions with a view to understanding the concept.

4.2.16. Experimenter generated examples

To help the subject understand a concept the experimenter may intervene to create new examples to illustrate the concept. Here is a further example from Anderson et al.'s protocol in which the experimenter presents a new 'top-level' example problem for the subject to think about:

E: Ok, let me try [?] this way. Suppose we had typed in open paren CAR of QUOTE paren LIST1 closed paren closed paren [(CAR '(LIST1))], what do you expect to get as an answer to that?
B: Paren LIST1.

Such experimenter generated examples are represented as in figure 3.42. The parallelogram is used to distinguish such examples from those in the textbook itself.

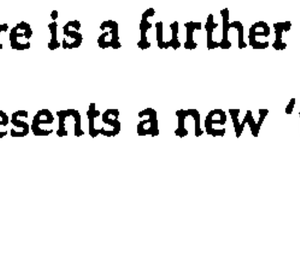


Figure 3.42. Experimenter generated example solution.

5. Applying the framework to a training manual: An analysis of Winston and Horn (1981) pp. 33-36.

To illustrate the use of the interpretation theory for a relatively coarse grained analysis of training problems, the following examples are taken from Winston and Horn (1981). Chapter 3 of the book (pages 33-36) deals with 'definitions, predicates, conditionals and scoping' in LISP. The first description of a function definition is given on page 33. It takes the form of a template showing the syntax of function definitions in general:

(DEFUN <function name>
    (<parameter 1> <parameter 2> ... <parameter n>)
    (<process description>)

The reader is reminded that the angle brackets 'delineate descriptions of things' and provides some examples of those things. There are three parts of the function definition which the reader has to understand: the first part contains the function name, which the programmer has to provide; the second part is a *parameter list*, which provides arguments for the third part: the *process description*. This last part is what the function actually does. Each part of the template is explained in the text which follows the template.

Figure 3.43 represents the fact that the template (B^t) is followed by an explanation of it (B^te). The B^te term represents a section of text in which the relationship between the syntax and the form of the template is explained. This relationship is represented by the solid line from the template B^t to the explanation B^te.

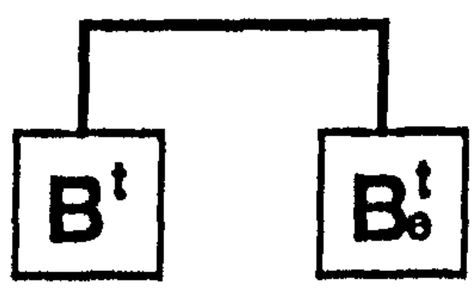


Figure 3.43. Representation of the relation between the template and its explanation.

No explicit links are thereafter made in the text between the examples provided and the template at the beginning of the section. Due to the nature of DEFUN, there is no one procedure which applies to all the examples provided, other than that of establishing a definition name and giving a list of parameters for the function to work on.

Example 1:

The template and its explanation are followed by an example problem (labelled A_1) and a possible solution to the problem (a piece of LISP code labelled B_1 below):

A1: '...define a function that converts temperature given in degrees Fahrenheit to degrees Celsius.'

B1: (DEFUN F-TO-C (TEMP)
(QUOTIENT (DIFFERENCE TEMP 32) 1.8))

The LISP code is followed by a partial explanation of how the program works (B1e). This explanation deals with how the *argument* is evaluated becoming the temporary value of the function's *parameter*. It does not say how the second line in the above code is evaluated (hence the minus sign above the 'e' in the B1e term in figure 3.44). There are only partial mappings between elements in the problem statement and the final solution. Some are explicitly given and others have to be inferred:

parameter ⇒ *TEMP* ⇒ *degrees Fahrenheit (inferred)*
value returned ⇒ *37.77* ⇒ *degrees Celsius (inferred)*

No further mention is made of 'converts', 'temperatures', 'degrees Fahrenheit' and 'degrees Celsius'. There is no explanation of how to construct the second line of the code: (QUOTIENT (DIFFERENCE 100 32) 1.8). Since there are only partial mappings between the terms (A1, B1 and B1e), a dotted line is used to link the terms in figure 3.44 showing that the reader has to make a number of inferences to understand the relations. The black box indicates that there is no explanation about how to generate the second line of code.

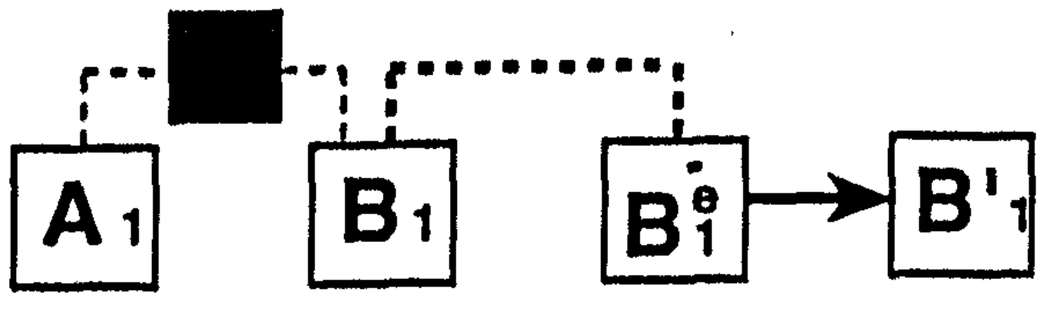


Figure 3.44 Relations between the statement of the first example problem and the solution and explanation.

Half-way through the explanation the reader is asked to refer to the figure on page 35 reproduced as figure 3.45.

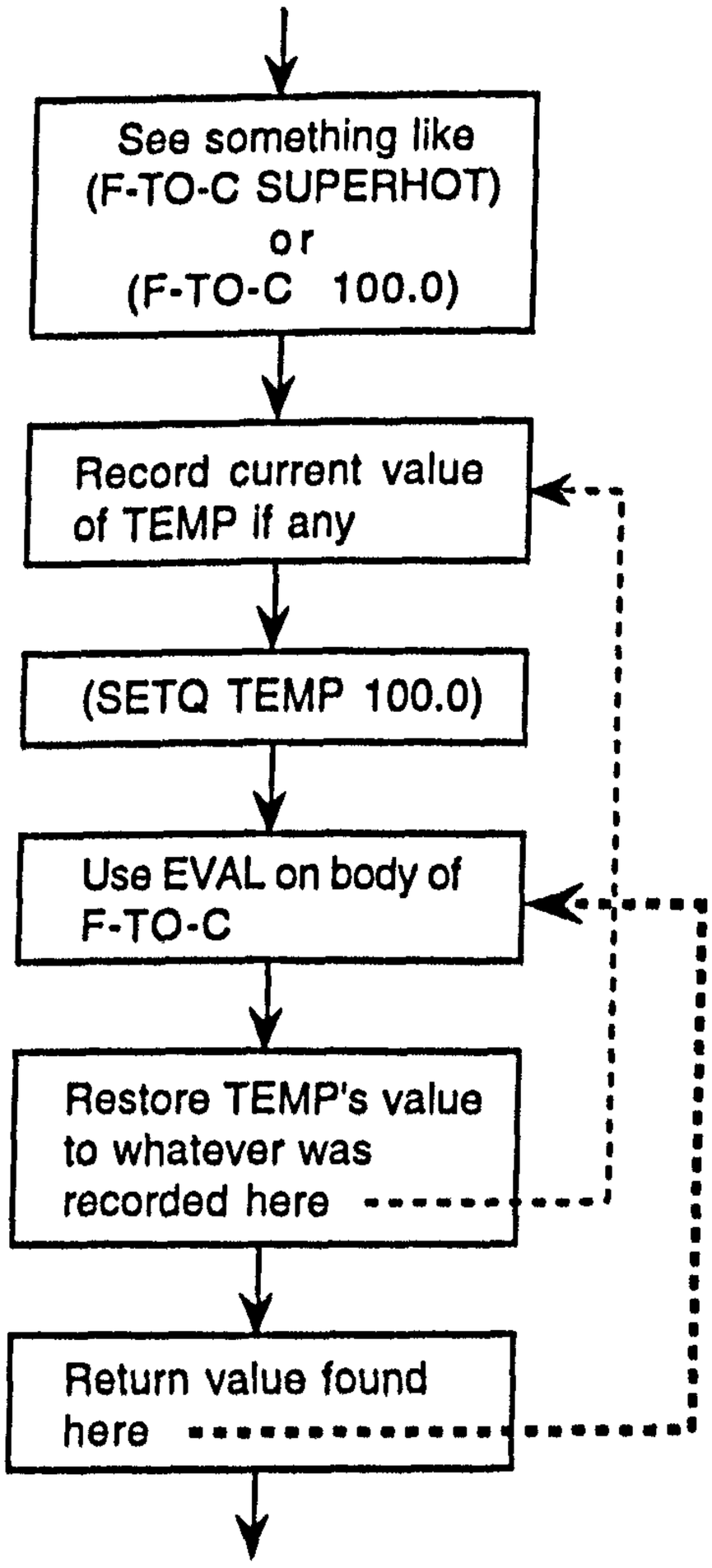


Figure 3.45. Representation of the order of operations carried out when example 1 is evaluated (from Winston and Horn, 1981, page 35)

This diagram is meant to illustrate what is happening as the program is being evaluated (represented by the arrow from B1e and B'1 in figure 3.44). The new, intermediate representation is designed to allow readers to re-represent the problem in order to help them understand how it works. However, this new representation of how the program works is not *entirely* analogous to the working of the actual program. In the diagram represented by B'1, the third box contains the LISP code: (SETQ TEMP 100.0), which is intended to show that the parameter TEMP is assigned the value 100.0 before the body of the code is evaluated. Yet the LISP form SETQ is not used in the code itself. The code works only *analogously* to SETQ.

Example 2:

The second example problem statement is also followed immediately by the solution and explanation:

A2: '[define] a function which exchanges the first and second elements of a two-element list.'

B2: (DEFUN EXCHANGE (PAIR)
(LIST (CADR PAIR) (CAR PAIR))) ;reverse elements

The explanation provided for this example is more complete than the previous one in that the mappings between elements in the problem statement and the form of the solution are given. The reader is asked to assume that the value of the variable SINNERS is the two-element list (ADAM EVE). The explanation goes on to explain what happens when (EXCHANGE SINNERS) is evaluated. In doing so it provides the following mappings:

SINNERS	⇒*PAIR*	
(CADR PAIR)	⇒*EVE*	⇒*2nd element of two-element list*
(CAR PAIR)	⇒*ADAM*	⇒*1st element of two-element list*

Once these have been explained the book then says that 'the following is evident:

(EXCHANGE SINNERS)
(EVE ADAM)'

However, the book does not explain the effect of CADR, CAR or LIST in this example. It assumes the reader already understands them since they were presented in the preceding chapter, hence the black box in figure 3.46. For this reason the B2e term in figure 3.46 has a minus sign above it.

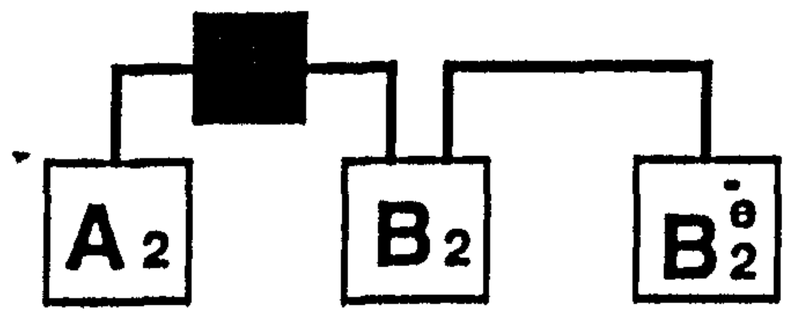


Figure 3.46. Representation of the third example 'EXCHANGE' from page 36 of Winston and Horn

Example 3:

A3: Define a function to compute the percentage by which the second argument given in the function is greater than the first.

B3: (DEFUN INCREASE (X Y)
(QUOTIENT (TIMES 100.0 (DIFFERENCE Y X)) X))

(INCREASE 10.0 15.0)
50.0

(INCREASE 10.0 15.0) is used as an example to show the mapping between the first and second arguments (10.0 and 15.0) and the X and Y of the solution (B_3). No explanation is provided of QUOTIENT, TIMES or DIFFERENCE, nor why Y precedes X in the second line, nor of the position of X at the end of the second line. The relation between statement and solution is therefore only partially explained. The relation between the following elements from the problem statement and the solution are given:

X $\Rightarrow$ 10.0
Y $\Rightarrow$ 15.0

Since there is no explanation of the operations required to derive the body of the code, the reader is left to infer the interrelations between QUOTIENT, TIMES and DIFFERENCE and the form the syntax takes. Because of the paucity of explanation and the number of inferences left for the reader to make, figure 3.47 shows a dotted line and black box between the A term and the B term.

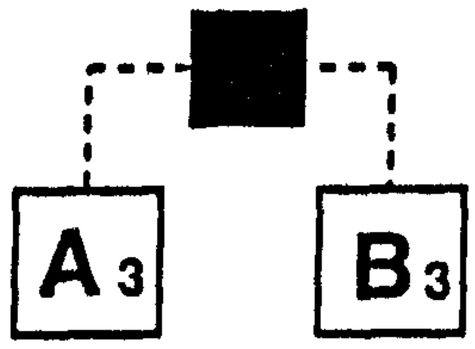


Figure 3.47. In the third example a number of relations between the statement and solution have to be inferred.

Example 4:

In this example the problem is the same as the first example on page 34. The form of the solution is different from the previous F-TO-C example, however. The explanation given after the solution concerns *side-effects*. There is no explanation about how to construct the code given the problem statement.

A_4: (problem is not stated. Reader has to refer back to previous example (A1))

B_4:
```
(DEFUN F-TO-C (TEMP)
        (SETQ TEMP (DIFFERENCE TEMP 32))
        (QUOTIENT TEMP 1.8))
```

B4e: There is only a partial explanation of the code: TEMP is used as both input parameter and temporary anchor for the difference between the input and 32. How the same variable can play two different roles is not explained.

Figure 3.48 shows that there is no relation provided between the problem statement and the solution (there is no line linking the A term and the B term), and the partial explanation of the B term is represented by the dotted line.

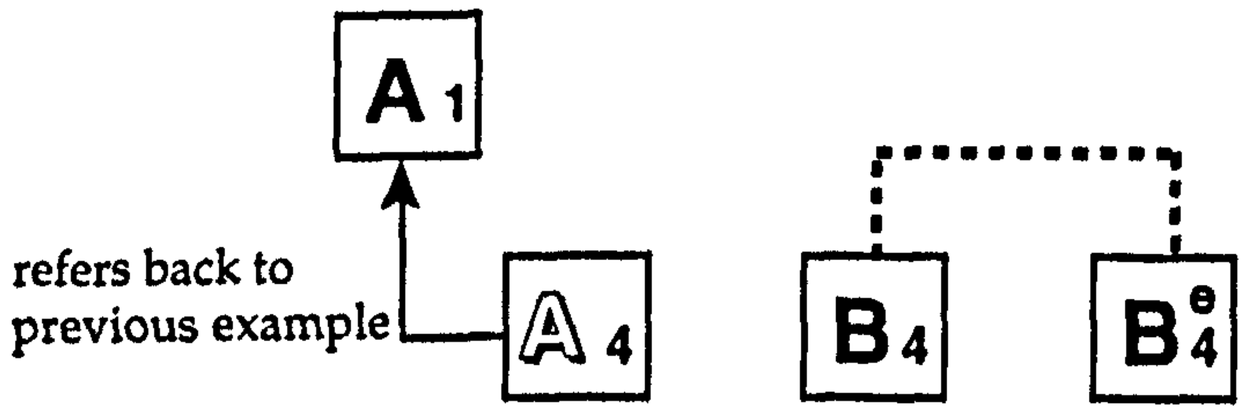
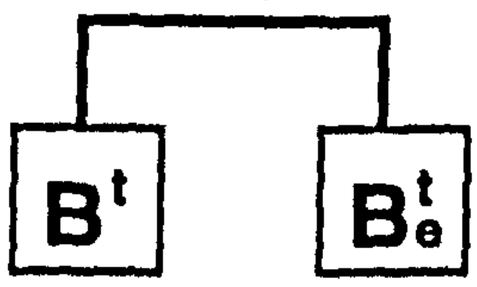


Figure 3.48. Representation of example 4 involving only a partial explanation of the solution.

Figure 3.49 summarizes the analysis of this section of the textbook. We can see at a glance what inferences the reader is required to make (the dotted lines) and therefore where the reader may have difficulty later. It is also apparent that there are no explicit links between the individual problems. That is, the writers do not explicitly relate one problem to another. They do not explain in what way, for example, F-TO-C relates to the EXCHANGE example. For this reason no lines are drawn linking one example to another.

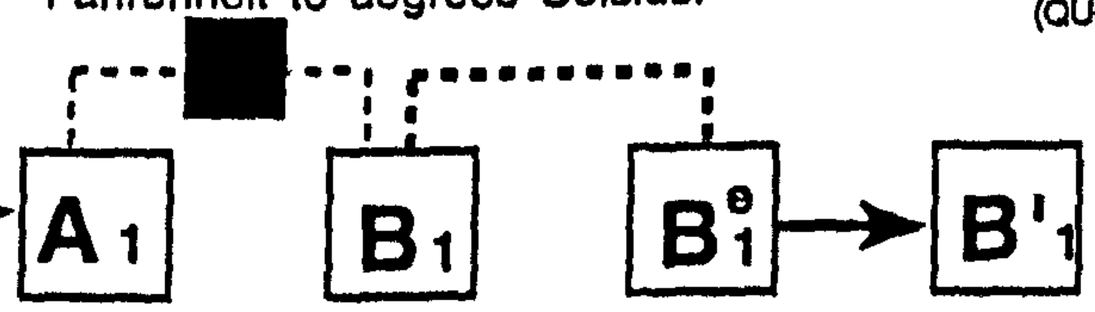
DEFUN template and explanation




```
(DEFUN <function name>
        (<parameter 1> <parameter 2> ... <parameter 3>)
        <process decription>)
```

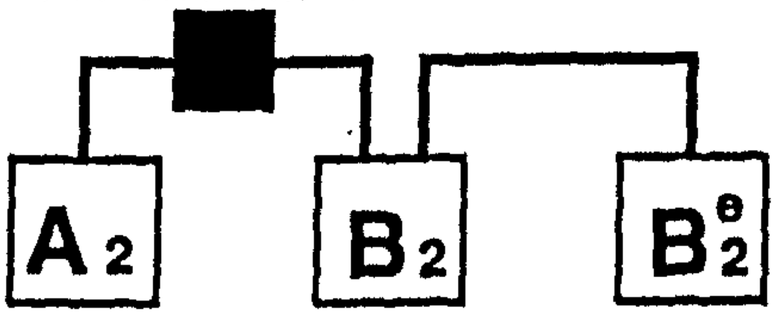
Example 1
'... define a function that converts temperatures given in degrees Fahrenheit to degrees Celsius.'

```
(DEFUN F-TO-C (TEMP)
        (QUOTIENT (DIFFERENCE TEMP 32) 1.8))
```



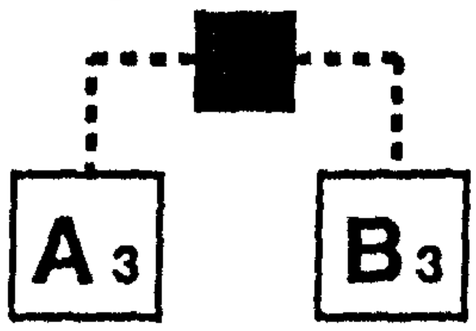
Example 2
'This new function exchanges the first and second elements of a two-element list'

```
(DEFUN EXCHANGE (PAIR)
        (LIST (CADR PAIR) (CAR PAIR)))
```

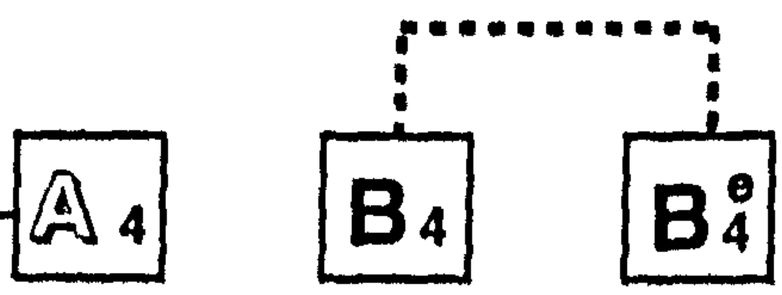


Example 3
'Its purpose is to compute percentage. More specifically, the value returned is to be the percentage by which the second argument given to the function is greater than the first.'

```
(DEFUN INCREASE (X Y)
        (QUOTIENT (TIMES 100.0
                (DIFFERENCE Y X)) X))
```



refers back to previous example




```
(DEFUN F-TO-C (TEMP)
        (SETQ TEMP (DIFFERENCE TEMP 32))
        (QUOTIENT TEMP 1.8))
```

Figure 3.49. Analysis of pages 33-36 of Winston and Horn (1981) showing the relations between problems and their solutions.

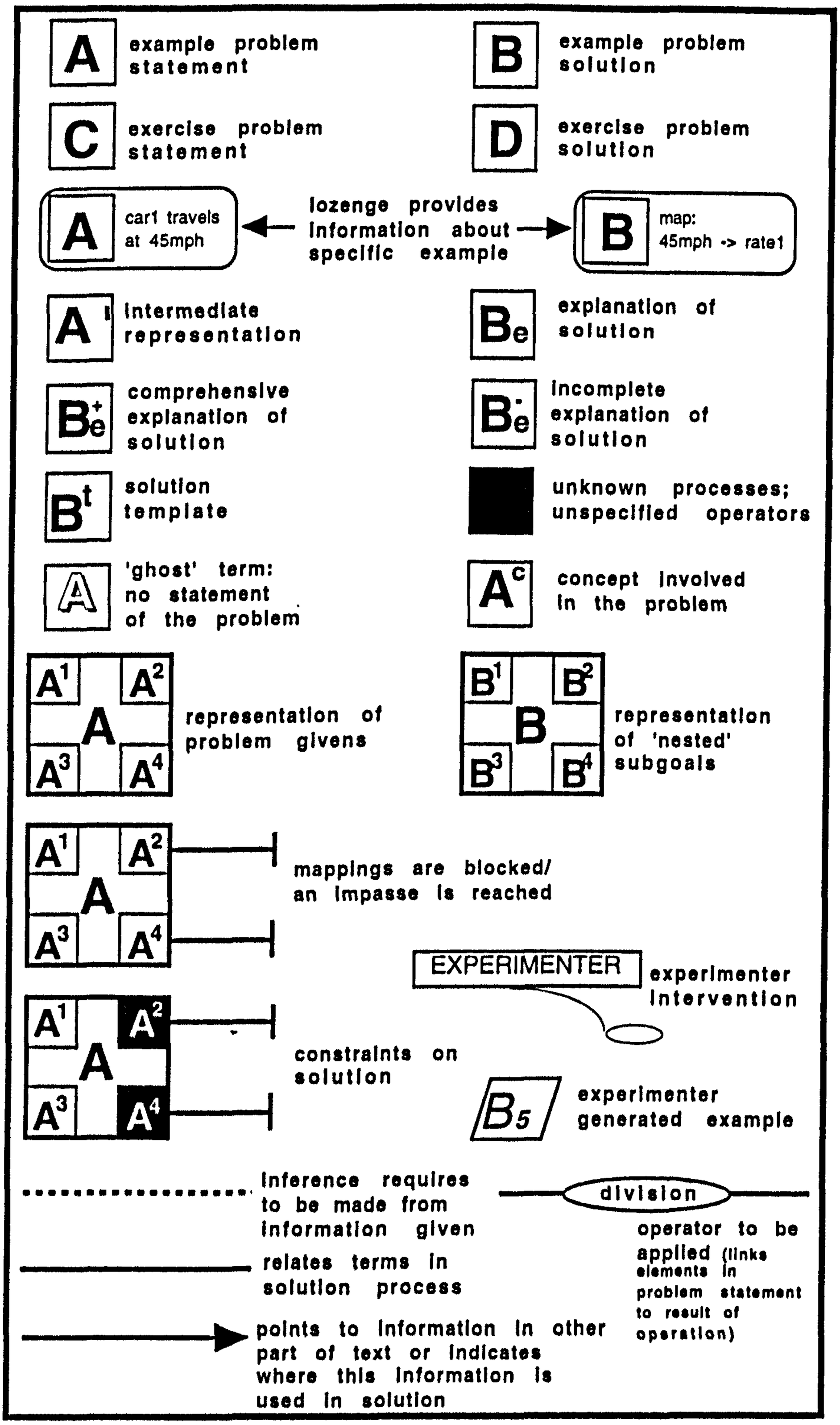


Figure 3.50. Summary of the notation used in the text and protocol analysis.

Chapter 4 TEXT AND TASK ANALYSIS USING THE INTERPRETATION THEORY

1. Interpretation theory applied to laboratory studies

In this chapter, the interpretation theory will be used to examine in depth the structure of a simple word problem that is described and the solution explained in Reed, Dempster & Ettinger (1985). The theory will also be used to analyse and predict the effects of those aspects of a problem explanation where inferences are required to solve another problem of the same type or a variant of the source problem.

1.1. The effect of having to make inferences

Britton, Van Dusen, Glynn, & Hemphill (1990) examined the effect of different types of inferencing in the study of instructional texts by 'naïve, unmotivated, passive readers'. Figure 4.1 represents a problem as a tree structure in which there are a number of nodes linked to other related nodes. These nodes represent points at which inferences are made and hence at which further information becomes available. If the hypothetical solver is able to make the two inferences enclosed in the squares in figure 4.1 then the rest of the tree structure becomes available and a solution can be found. If, as in figure 4.2, solvers cannot make the necessary inferences at those points, then they will be unable to relate one part of the text to another.

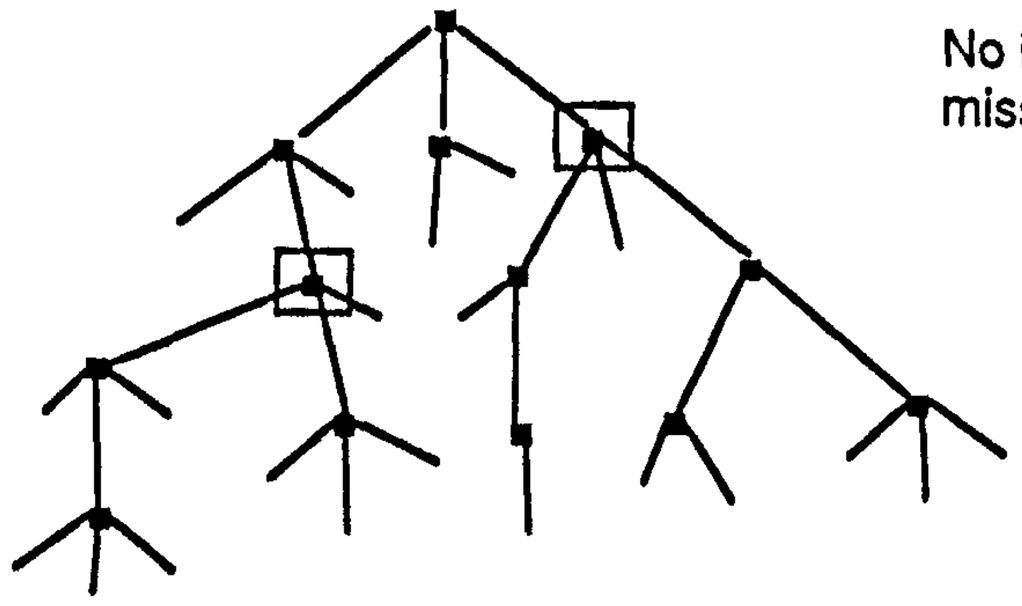


Figure 4.1. Problem solving as a tree structure with nodes representing points at which inferences are made. From Britton, et al. (1990).

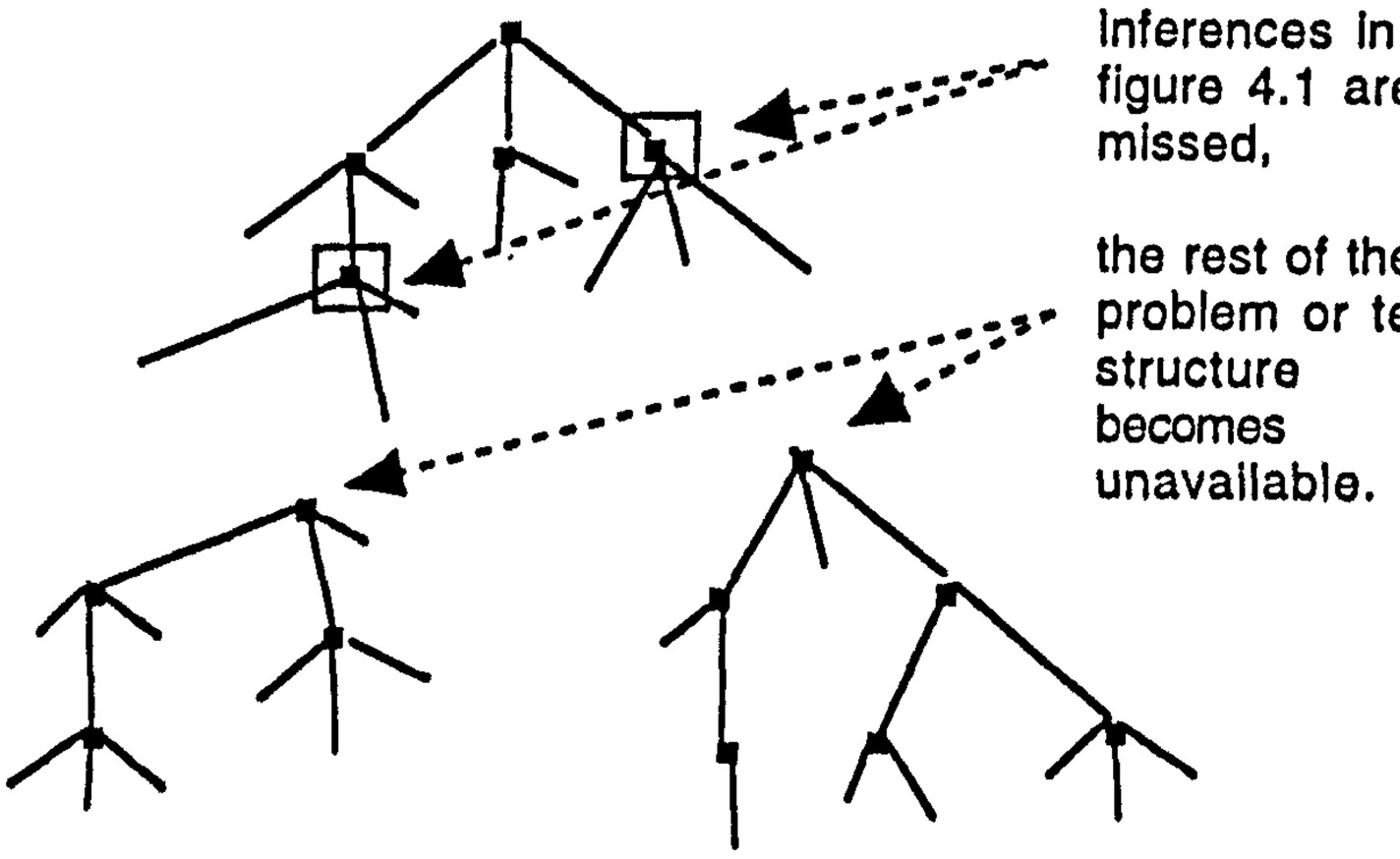


Figure 4.2. Representation of difficulties facing solver when two inferences are missed.

There are many kinds of inference involved in text comprehension. In word problems readers have to construct ã situation model of what the story represents. This model contains inferences that are derived from a reader's knowledge of the world. For example, if two cars leave the same place at different times, travel the same route, and the second car eventually overtakes the first, then readers might be expected to infer that both cars travel the same distance. Such *text-external* inferences require the reader to have some prior knowledge from outside the text. In the case of textbooks, writers also have to judge what the reader might be expected to remember from earlier parts of the textbook - that is, what *text-reinstatement* inferences readers are able to make - when they come to do a problem in later sections. The algebra problems examined in this section are very short and do not involve text reinstatement inferences.

2. Algebra word problems

A common type of algebra word problems is one where a vehicle travels from A to B at a certain speed and in a certain time ('linear distance' problems). A subtype of that class of problems involves 2 vehicles travelling the same route and leaving at different times ('colinear distance' problems). In both, the solver is required to generate the correct equation and map the values in the problem statement onto the variables in the equation, and then solve it for any unknown variable. In the next subsection I shall describe linear problems and in the following subsections colinear problems will be discussed.

2.1. Linear word problems

Distance problems employ variations on the basic equation 'distance = rate x time'. Since this type of problem has three variables - distance, rate, and time - three different equations can be constructed:

type 1 distance = rate x time
type 2 rate = distance ÷ time
type 3 time = distance ÷ rate

This class of equations has the same simple overall structure. The only difference between them is in the nature of the arithmetic operator needed to solve the problems - either multiplication (problem type 1) or division (problem types 2 and 3). They also involve problem statements which are easy to map onto the equations. In mathematics textbooks they are usually represented in a triangle where D is Distance, S is Speed and T is Time. The D can be found by multiplying S and T and the S and the T can be found by dividing D by T and S respectively:

Here are examples for each type of equation:

Type 1:

'A car travels at a speed of 45mph for 4 hrs. How far does it travel?'

Distance = rate x time

'45mph' maps onto 'rate' and '4 hrs' maps onto 'time'. Applying the multiplication operator to them gives the solution.

Type 2:

'A car travels 180 miles in 4 hrs. How fast does it travel?'

Rate = distance ÷ time

'180 miles' maps onto distance and '4 hrs' onto time. Dividing yields the speed.

Type 3:

'A car travels 180 miles at 45 mph. For how long does it travel?'

Time = distance ÷ rate

One point to note is that there is nothing explicit in the problem statement to indicate which operator to apply.

2.1.1. Explanations in mathematics textbooks

In colinear word problems prior knowledge is required about the concepts of rate, time, distance, the equality of the distances travelled, the differences in departure times, the mathematical operators that relate distance to rate and time, and those required to solve an alegbraic equation. When the simpler linear problems are first introduced in textbooks, they often take the following form:

Miss Matthews has to drive 450km from Newcastle to London for a conference. She thinks she can average 65 km/h. How long will she take?

65 km ⟷ 1 hour

450 km ⟷ $1 \times \frac{450}{65}$ hours

= 6.92 hours

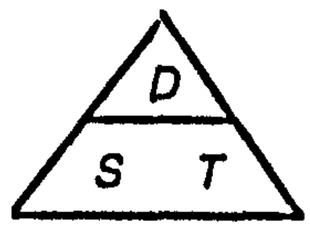
= 6 hours 55 minutes

Time = $\frac{\text{Distance}}{\text{Speed}}$

= $\frac{450}{65}$

= 6.92 hours

= 6 hours 55 minutes



Formula

$T = \frac{D}{S}$

(0.92 x 60 = 55.2)

Figure 4.3. Introduction to linear distance problems in Howat, Mullan, Nisbet & Brown (1987).

This example is taken from Howat, Mullan, Nisbet & Brown (1987) page 27. Similar types of explanation can be found in Holderness (1987) and Rayner (1990). In the example on the left of the figure there is no explicit explanation of the relation between the problem

givens, 65 km and 450 km, and the figures on their immediate right; nor is there a stated relation between them and the concepts in the centre of the figure; nor between either of those and the triangle and formula given; nor what the figures in brackets have to do with anything. The explanation given in figure 4.3 is represented in figure 4.4. In row *a*, time is related to distance and speed but none of these is explicitly related to the problem statement.

The reader has to infer that 450km and 65km/h should map onto distance and speed (rows b and c) in the equation - hence the dotted lines from the problem givens A^1 and A^2 to the subgoals B^1 and B^2. There is no explanation why the equation is in the form it is (row d). The reader has to infer that the speeds and distances in B^1 and B^2 should map onto the equation in row *d*. There is no explanation of what operations to perform to get the figure 6.92 hours (row e). There is no explanation why .92 is multiplied by 60 (B^5e in row f), nor how this relates to the solution (B^5 in row f). In row *g*, the 5 subgoals (B^1 to B^5) are shown as 'nested' within the B term.

The student is then asked to solve 8 problems based on this example. Unless the student has some expertise in mathematics, these problems may prove rather difficult since the example does not contain any explanations. Indeed, figure 4.3 contains a bewildering collection of unrelated numbers and equations that make sense only if the reader already understands what the example is trying to explain.

This example highlights the fact that some textbooks in widespread use should and can be readily improved.

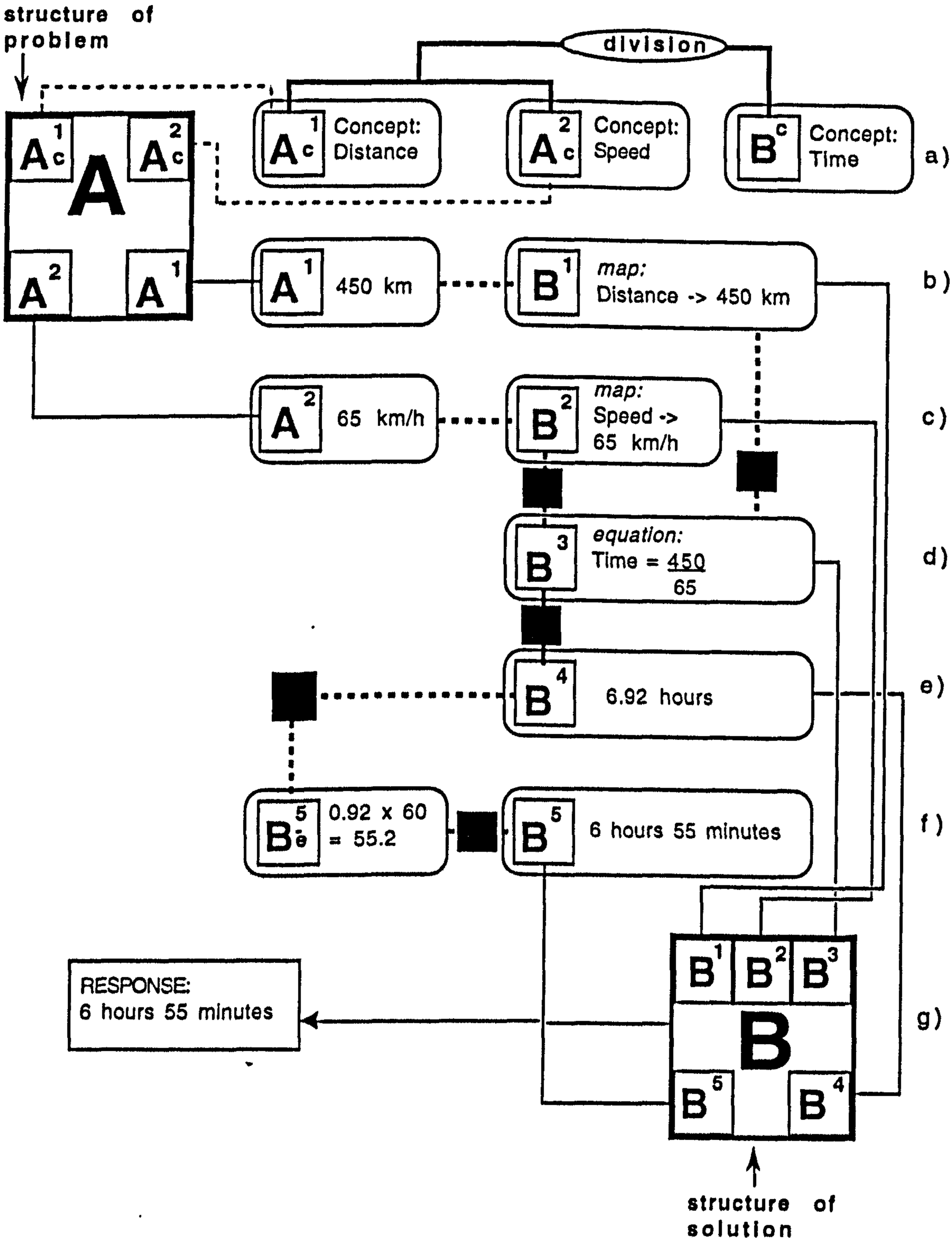


Figure 4.4. Analysis of the example problem on speeds in Howat et al. (1987).

2.2. Colinear word problems

In colinear problems, 2 vehicles travel at different speeds over the same distance. They therefore involve a variant of the first type of basic *Distance = Rate x Time* problems. Since the distance travelled by each vehicle is the same, the *Rate x Time* for each vehicle

must also be the same. The relevant equation then becomes $Rate_{vehicle1} \times Time_{vehicle1} = Rate_{vehicle2} \times Time_{vehicle2}$.

2.2.1. Solutions to colinear problems

In colinear problems the textbook writer or researcher can decide which variable is unknown. In the following example $Time_{vehicle2}$ is unknown.

Truck1 leaves point A at 45mph.
Truck2 leaves point A at 55mph
Truck2 leaves point A 2 hours after Truck1.
How long does it take Truck2 to catch up with Truck1?

There are several ways to solve this problem:

a) A solver might calculate where each truck is at each succeeding hour until both trucks are at the same distance from point A. Truck1 will travel 45 miles in 1 hour, 90 miles in two hours, 135 miles in three hours, and so on. 2 hours after truck1 leaves, truck2 travels 55 miles in each succeeding hour. Working it out like this, both trucks will meet after 9 hours, having travelled 495 miles.

b) A solver might reason that, since truck1 has travelled 90 miles in the first 2 hours, truck2 has to make up this 90 miles travelling 10mph faster. Since it will gain 10 miles each hour, it will take the truck 9 hours to catch up.

c) A solver can use the algebraic equation $Rate_{vehicle1} \times Time_{vehicle1} = Rate_{vehicle2} \times Time_{vehicle2}$.

2.2.2. The algebraic solution

Unlike the linear problems, where there are three variables, one of which is unknown, colinear problems have 4 variables but only two of them, in this case the rates, can be instantiated from the values given in the problem. The algebraic solution depends on the solver realizing that both vehicles travel the same distance, and that the first vehicle travels for the same length of time as the second vehicle *plus* the length of time it was on the road before the second one left. So the time for vehicle1 can be stated in terms of the time taken by vehicle2.

$$\text{Time}_{\text{vehicle1}} = \text{Time}_{\text{vehicle2}} + d$$

where *d* is the difference in departure times. The complete equation for solving the type of problem where *time* is the unknown looks like this:

$$\text{Rate}_{\text{vehicle1}} \times (\text{Time}_{\text{vehicle2}} + d) = \text{Rate}_{\text{vehicle2}} \times \text{Time}_{\text{vehicle2}}.$$

The equation for the problem in section 2.2.1 becomes:

$$45 \times (\text{Time}_{\text{vehicle2}} + 2) = 55 \times \text{Time}_{\text{vehicle2}}$$

From this point on the solution depends on the solver's ability to select the relevant arithmetic operators in order to solve the equation. They are:

Step 1. *Multiplication* of terms in brackets.

$$45 \times \text{Time}_{\text{vehicle2}} + 90 = 55 \times \text{Time}_{\text{vehicle2}}$$

Step 2. Put terms of same type on same side of equation -*subtraction.*

$$90 = 55 \times \text{Time}_{\text{vehicle2}} - 45 \times \text{Time}_{\text{vehicle2}}$$

Step 3. *Subtraction* of terms on right-hand side;

$$90 = 10\ \text{Time}_{\text{vehicle2}}$$

Step 4. Simplify (by *division*).

Divide both sides by 10 to find the '$\text{Time}_{\text{vehicle2}}$';

$$9 = \text{Time}_{\text{vehicle2}}$$

Step 5. Replace units.

Since time is given in hours and minutes, then:

Time taken by vehicle2 is 9 hours

2.2.3. Mappings and operations in a colinear problem where *rate* is unknown

Colinear problems in which the *rate* is unknown can have either a similar or more complicated algebraic structure than those in which *time* is the unknown. In the more complicated structure, only one value in the problem statement can be directly mapped onto the equation, as in the following example:

> *Car1 leaves point A.;*
> *2 hours later car2 leaves point A travelling 10mph faster than car1 and overtakes it after 6 hours;*
> *At what rate must the second car travel to overtake the first?*

In this case there is not only a difference in departure times but also a difference in rates between the two cars. The difference in rates has to be added to the equation along with the difference in times. In this case this means that $Rate_{car1} = Rate_{car2} - 10$ has to be substituted for $Rate_{car1}$ in the equation. The full equation now looks like this:

$$(Rate_{vehicle2} - D) \times (Time_{vehicle2} + d) = Rate_{vehicle2} \times Time_{vehicle2.}$$

where D is the difference in rates and *d* is the difference in times.

3. Task analysis of the Reed, Dempster & Ettinger (1985) practice distance problem

Reed, Dempster & Ettinger (1985) used variations on the above colinear distance problems in their studies of transfer. The subjects in their experiments were novices and had never solved algebra word problems before, although they had been taught some complicated algebraic manipulations. However, one of the difficulties in analyzing tasks in the detail presented here is finding that state of ignorance which allows us to identify where inferences are built into a text and therefore where novices are likely to encounter problems. For this reason no inference is considered trivial.

To spell out in some detail the operations required to solve one of Reed et al.'s problems, the interpretation theory will be applied first of all to a task analysis of the 'practice distance problem'.

3.1. The task analysis

Below is a statement of the practice problem given to the subjects in Reed et al.'s first three experiments.

> A car travelling at a speed of 30 miles per hour (mph) left a certain place at 10.00 a.m. At 11.30 a.m. another car departed from the same place at 40 mph and travelled the same route. In how many hours will the second car overtake the first car?

In order to present a task analysis of this problem we will extract the salient features:

Car1 travels at a rate of 30mph
Car2 travels at a rate of 40mph
Car1 leaves point A at 10.00 a.m.
Car2 leaves point A at 11.30 a.m.
Both cars travel from point A to point B
Both cars reach point B at the same time
Time taken by car2 to reach point B is ?

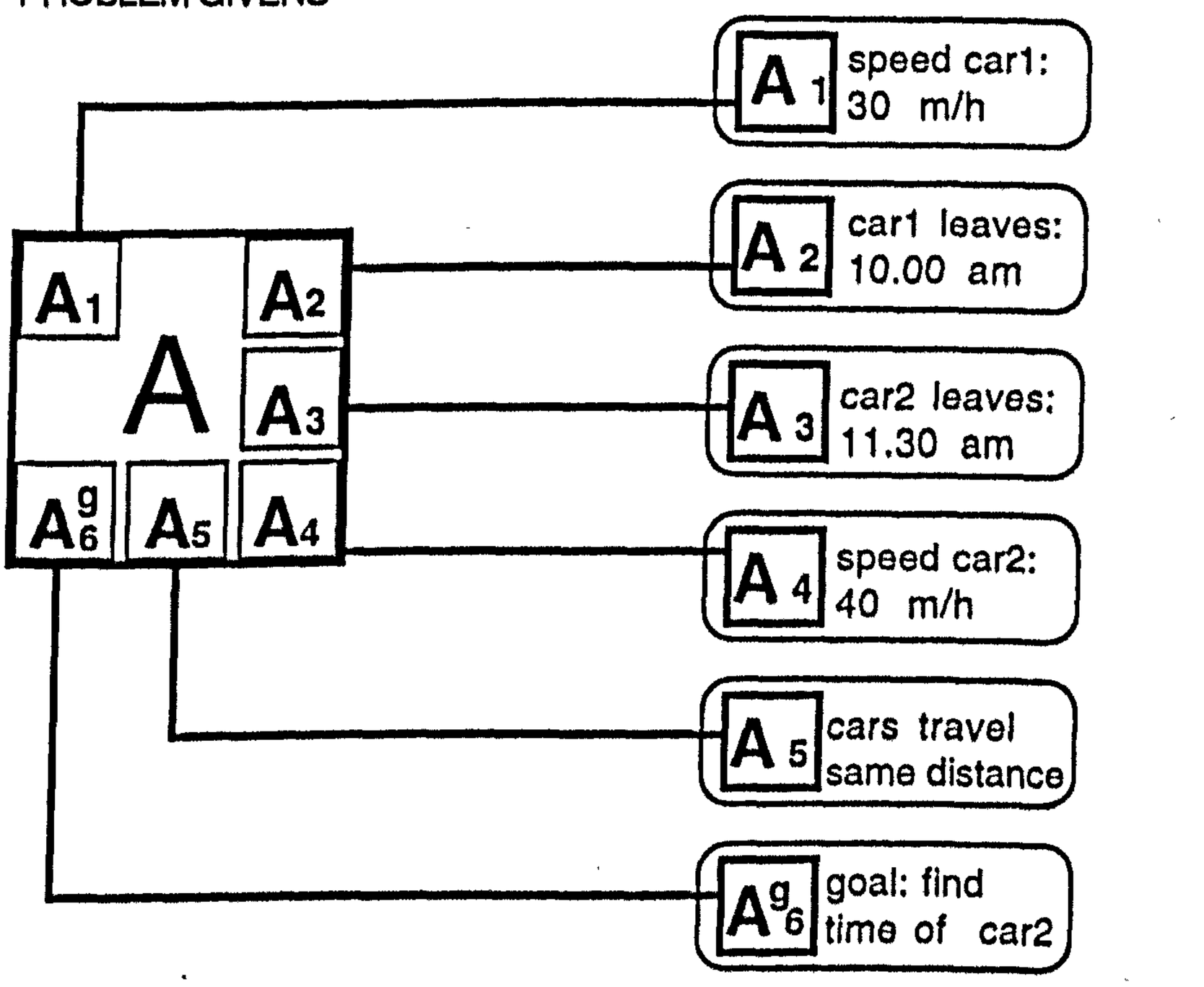


Figure 4.5. Representation of problem givens and concepts.

On the way to an overall solution, a number of subgoals have to be attained. These subgoals or partial solutions are then often used to achieve further subgoals. The analysis presented below is therefore divided into a number of steps each of which is in three parts. On the left-hand side of the page there is a natural language explanation of the subgoal to be achieved at each step and the operator involved. On the right-hand side there is a list showing the problem elements that have to be manipulated. These elements are either the problem givens or the results derived from a previous step in the solution (indicated by GIVEN and DERIVED respectively). These are followed by a statement of the *subgoal* of the particular step, the *operator* to be applied, and the *result* of that operation. The third part of each step is the interpretation theory representation of the givens (or the derived results from a previous step), the operator, and the result.

The A terms which appear in the remaining figures in this section refer to the givens and concepts in figure 4.5. The B terms are those which are derived from the problem givens; that is, they are the results of a previous step in the solution.

The first step identifies the general problem category in terms of the concepts involved: those of *distance, rate* and *time*; and how they are related:

1	The problem is a distance-rate-time problem in which two vehicles travel the same distance. If you know the rate and time of a vehicle then you can find the *Distance* it travelled by *multiplying Rate* and *Time*	*GIVEN: concept: rate* *GIVEN: concept: time* *SUBGOAL: find concept: distance* *OPERATOR -> multiplication* *RESULT -> concept: distance*

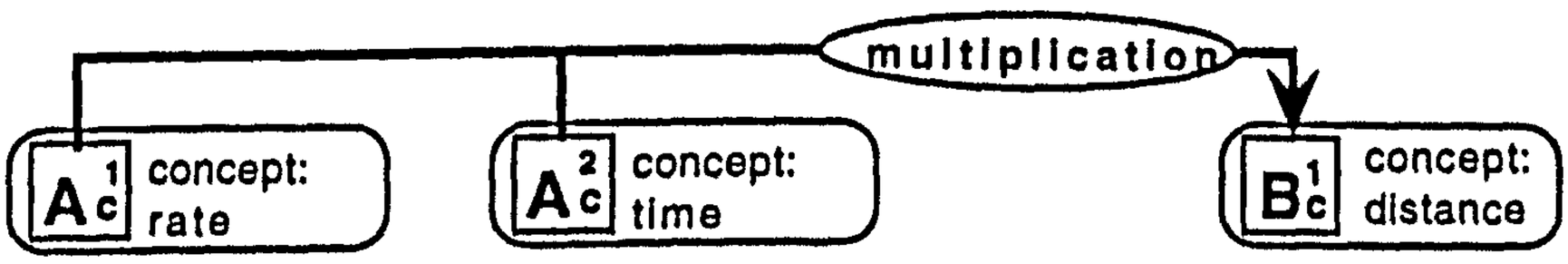


Figure 4.6. Concepts of rate and time related to distance through multiplication operator.

The second step identifies the problem as a member of a subordinate category of distance-rate-time problems in which there are two vehicles.

2	In this problem there are two vehicles so the equation for each vehicle would be: distance = rate$_{vehicle1}$ x time$_{vehicle1}$ and distance = rate$_{vehicle2}$ x time$_{vehicle2}$	*GIVEN: rate$_{vehicle1}$* *GIVEN: time$_{vehicle1}$* *GIVEN: rate$_{vehicle2}$* *GIVEN: time$_{vehicle2}$* *SUBGOAL: find distance$_{vehicle1}$* *OPERATOR -> multiplication* *RESULT -> distance$_{vehicle1}$* *SUBGOAL: find distance$_{vehicle2}$* *OPERATOR -> multiplication* *RESULT -> distance$_{vehicle2}$*

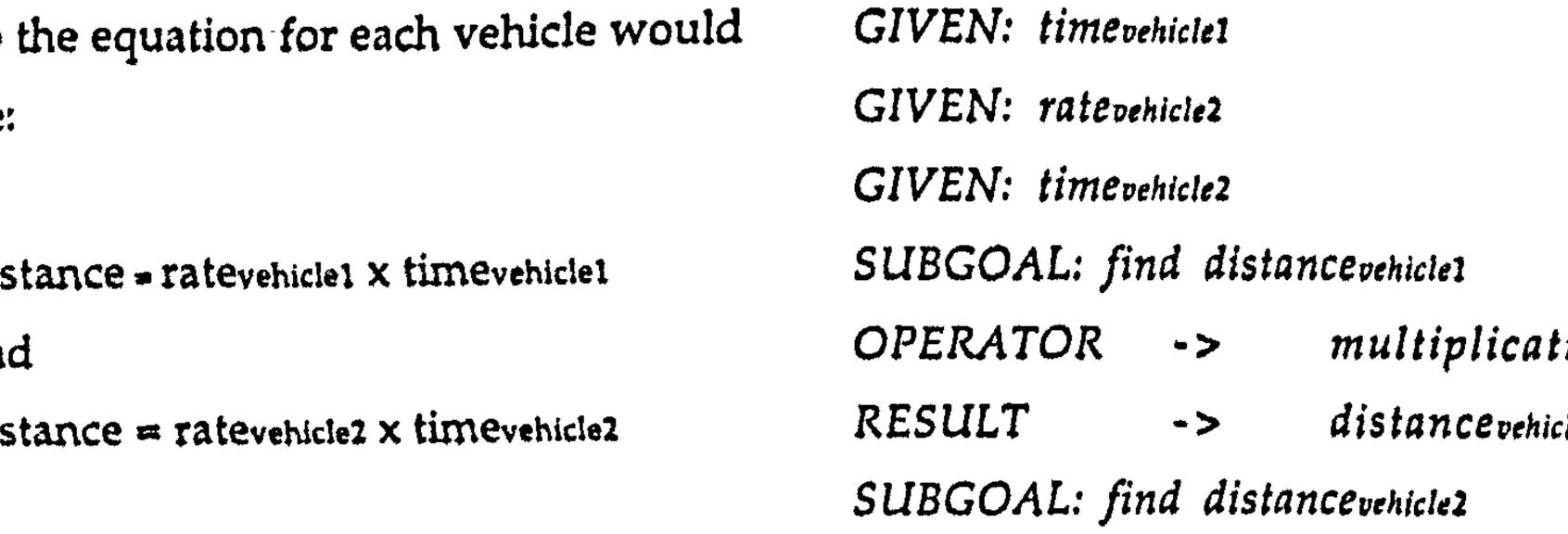


Figure 4.7. Concept of distance related to rate and time for two vehicles.

Step 3 shows how the relevant equation can now be derived from the fact that both vehicles travel the same distance.

3	Since the two vehicles travel the same distance, the distance travelled by one vehicle is equal to the distance travelled by the other. If the distances are equal, the *rate x time* for each vehicle must be equal too, so the equation when there are two vehicles travelling the same distance is: $rate_{vehicle1} \times time_{vehicle1} =$ $rate_{vehicle2} \times time_{vehicle2}$	*DERIVED: distance*$_{vehicle1}$ *DERIVED: distance*$_{vehicle2}$ *SUBGOAL: set distances equal to each other* *OPERATOR -> substitution* *RESULT ->* $rate_{vehicle1} \times time_{vehicle1} =$ $rate_{vehicle2} \times time_{vehicle2}$

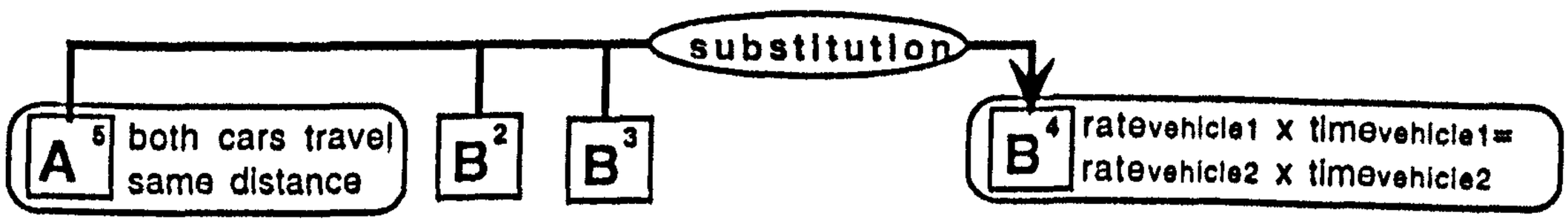


Figure 4.8. Substitution operator replacing *distance* with *rate x time*.

The first three steps above represent a general procedure for deriving the particular equation for this type of distance problem. The steps from 4 onwards show how the givens in this specific example map onto that equation.

4	We can represent whatever we want to find by a variable, such as t for the time.	*GIVEN: problem goal* *SUBGOAL: express goal as variable* *OPERATOR -> generate variable name* *RESULT -> t*

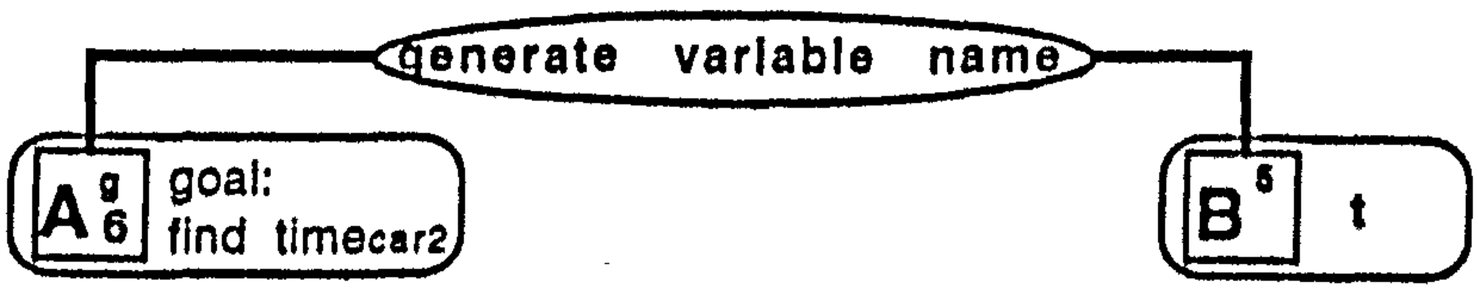


Figure 4.9. 'Generate variable name' operator.

Steps 5 and 6 involve finding any differences (in times) between the two vehicles and stating the time of one vehicle in terms of the other.

5 The slower vehicle leaves earlier and takes longer than the faster one. Calculate the difference between the time taken by the faster vehicle and the slower one:
11.30 - 10.00 = 1 1/2 hours

GIVEN: departure time car1
GIVEN: departure time car2
SUBGOAL: difference between times
OPERATOR -> subtraction
RESULT -> 1 1/2 hours

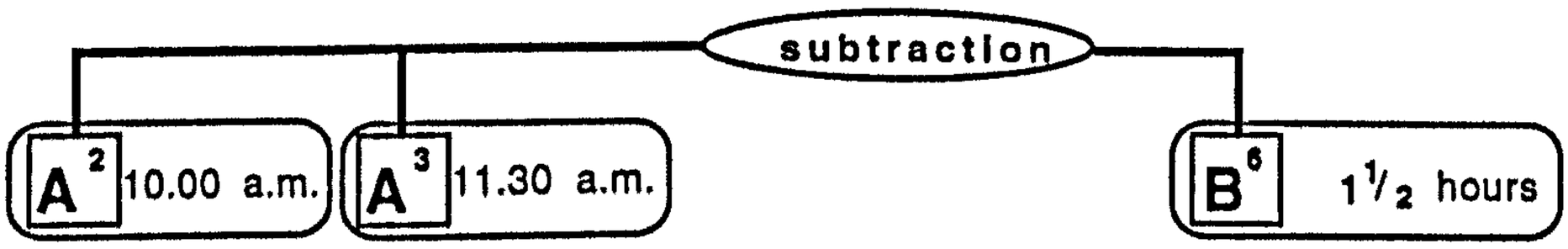


Figure 4.10. Subtraction operator applied to two times.

6 The slower car takes the same length of time as the faster one *plus* the difference in times between them.

DERIVED: time taken by car2 (t)
DERIVED: difference in times taken
SUBGOAL: time taken by car1 in terms of time taken by car2
OPERATOR -> addition
RESULT -> t + 1 1/2

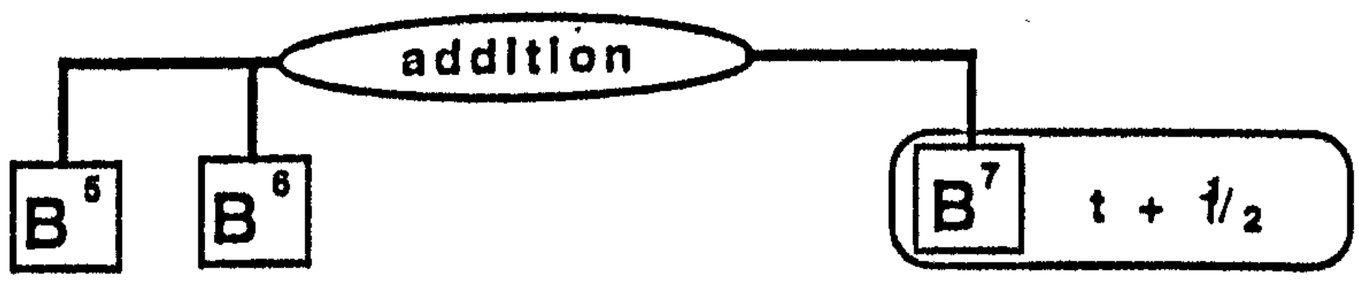


Figure 4.11. Addition operator applied to results of previous steps.

7 The variable slots in the equation can now be filled in with the values taken either from the problem statement or from the results of previous steps. The problem gives the rates of travel and the times taken have been derived in steps 4, 5 and 6.
('⇒' means 'maps onto')

GIVEN: rate$_{car1}$ ⇒ 30 m/h
GIVEN: rate$_{car2}$ ⇒ 40 m/h
DERIVED: time$_{car1}$ ⇒ t + 1 1/2
DERIVED: time$_{car2}$ ⇒ t
SUBGOAL: replace terms of equation with values
OPERATOR -> mapping
RESULT -> 30 x t + 1 1/2 = 40 x t

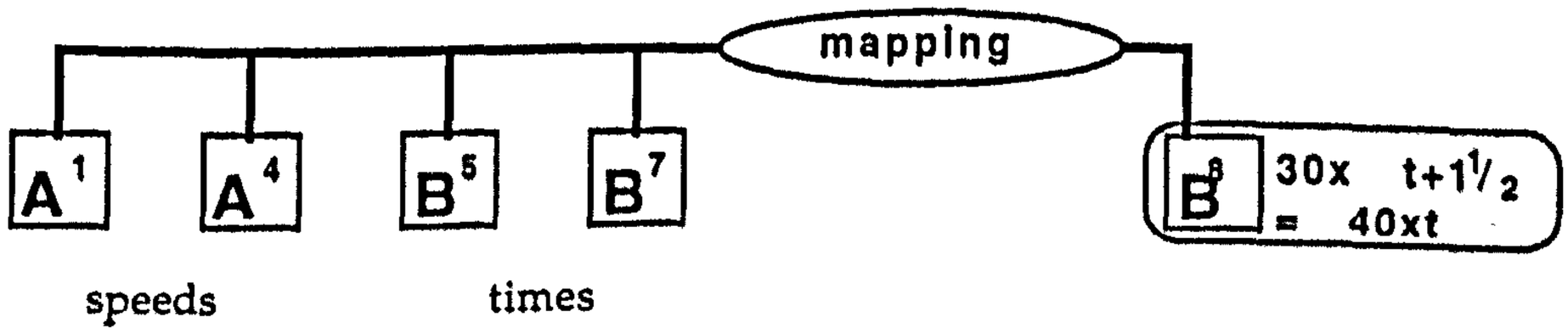


Figure 4.12. Mapping operator applied to values in table.

8 *If any of the terms have addition or subtraction signs then put brackets round them.*

DERIVED: t + 1½

SUBGOAL: put brackets round terms involving addition or subtraction

OPERATOR -> add brackets

RESULT -> (t + 1½)

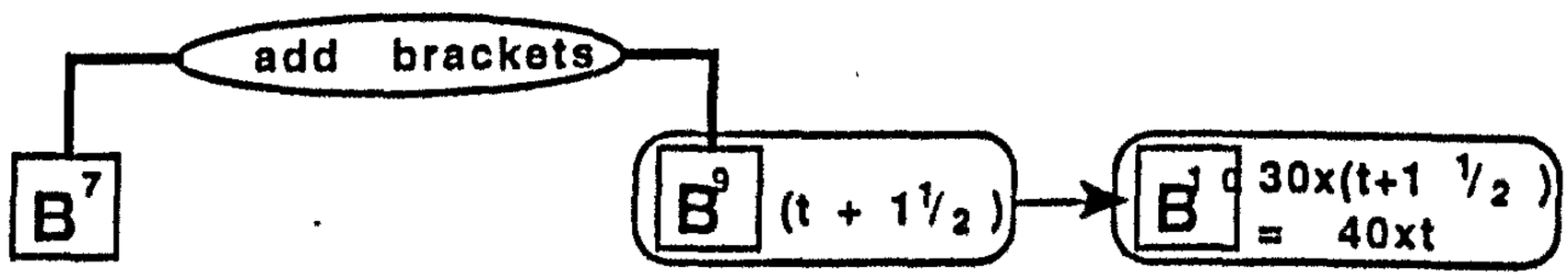


Figure 4.13. 'Add brackets' operator applied to elements.

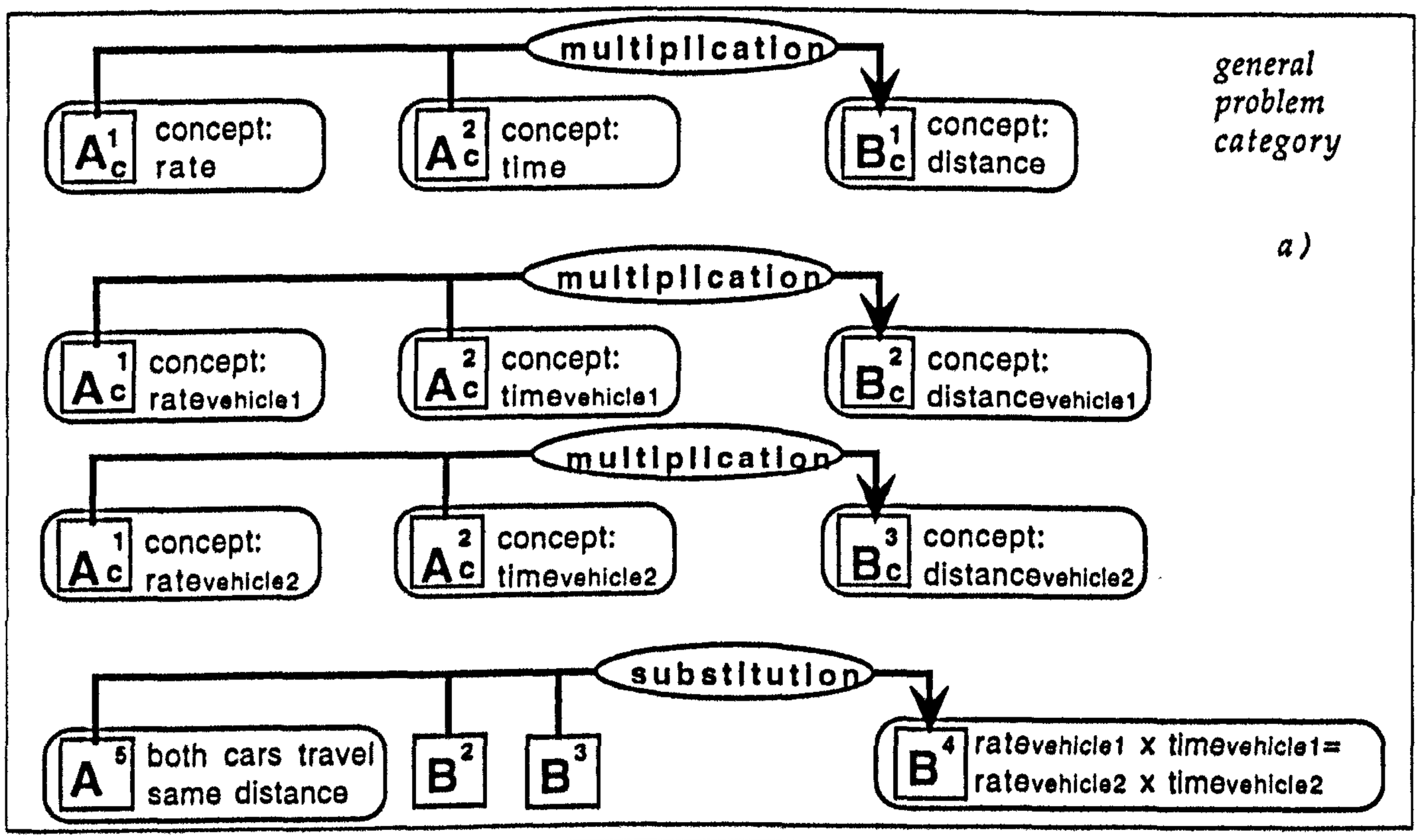
Step 9 involves applying a number of arithmetic operators to both sides of the equation. These are given in row *c* in the summary figure 4.14.

9 Solve for the unknown variable (t).

DERIVED: 30 x (t + 1½) = 40 x t

SUBGOAL: find value of t

In the study of these algebra word problems, the focus is on the solver's ability to generate the required equation. That is, researchers are interested in seeing if the solver can get as far as step 8 above, and what difficulties face the solver on the way. The ability of the solver to solve the equation once it has been generated is another question.



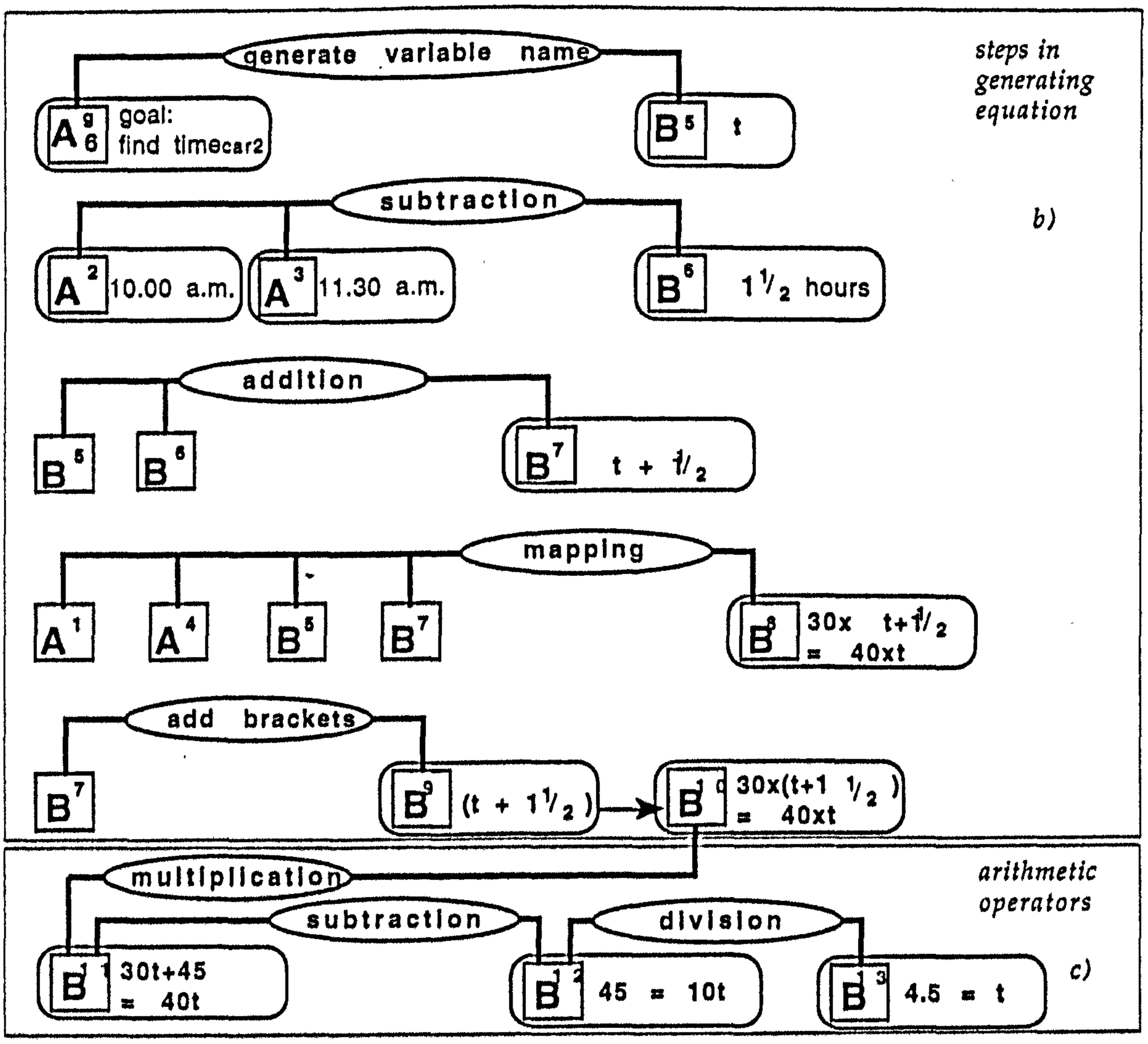


Figure 4.14. Summary figure of task analysis of practice distance problem.

4. The interpretation theory applied to laboratory experiments

This section will present a coarse-grained view of the experiments conducted by Reed et al. It is intended to illustrate the experimental design they used and how this design is likely to help or hinder the solver. In section 5, one of the practice problems will be looked at in detail followed by some of the test problems. We might then be able to see what aspects of the text structure might lead solvers to fail to solve the test problems when they refer to the practice problems.

4.1. The colinear problems studied by Reed, et al. (1985)

Reed, Dempster and Ettinger (1985) describe 4 experiments in which one example problem and solution is presented and the student is thereafter expected to solve a transfer problem, or a problem whose solution procedure was unrelated to the practice problem. In Reed et al.'s terminology the transfer problems were called 'equivalent' or 'similar'. Just how 'equivalent' and how 'similar' the problems really were will be the subject of later sections. In the mean time we will look at the experiments in general and at some of the algebra word problems in particular with a view to discovering just what the solution explanations that were provided *failed* to explain.

4.1.1. Overview of the experiments

Reed et al. classified the colinear problems as 'distance' problems which were related by the equation *Distance = Rate x Time*. The test problems were either a) 'unrelated' to the practice problems, that is, they did not use the same equation for their solution nor did they have the same surface features; or b) 'related', in that the problems required the same equation for their solution. The cover story for the related practice problem involved two cars travelling in the same direction at different speeds. An equivalent test problem was a close variant of the practice problem. Both involved cars leaving at different times and travelling at different speeds where the time taken by the vehicles is unknown. The similar test problem was a distant variant. The cover story for the distant variant involved trucks in place of cars and the requirement to find two rates rather than the time taken by one vehicle (this means that students, trained on a problem in which they have to find the Time, are obliged to find the answer to a *different problem*). The cover story for the unrelated practice problem involved batting averages for a baseball team in terms of the slugger and pitcher.

Two forms of solution explanation were provided in the experiments: an 'unelaborated' solution provided only some of the mappings between the example problem and solution leaving the subjects to make a large number of inferences. The 'elaborated' explanation provided more detailed mappings as well as an intermediate representation of the problem (a table).

4.2. The first 3 experiments

In the experiments the subjects had 6 minutes to solve each practice problem. They were then given the solutions and allowed to study them for two minutes. They were told that the solutions would help them solve later problems. In experiment 1, the practice problems and solutions were then removed and the subjects were asked to solve the test problems. In the second experiment, one group of subjects was allowed to consult the solutions to the earlier related problems and one group was not. In experiment 3, elaborated solutions to the practice problems were presented. Once again, one group was allowed to consult the practice problem, one group had the related problem removed before solving the test problem, and one group was given an unrelated practice problem to solve, which was also then removed.

4.2.1. Solving the related problem from memory

Figure 4.15 shows the relation between the current problem and the related practice problem. The relation between the A term and the B term is governed by the equation *Rate1 x Time1 = Rate2 x Time2* and relates all the distance problems presented in the experiments. However, in this condition neither the equation nor the explanation is available, so the student has to rely on memory, hence the shading covering the A and B terms, which represents the fact that we cannot be sure if the student has a representation of the source (A:B) in LTM.

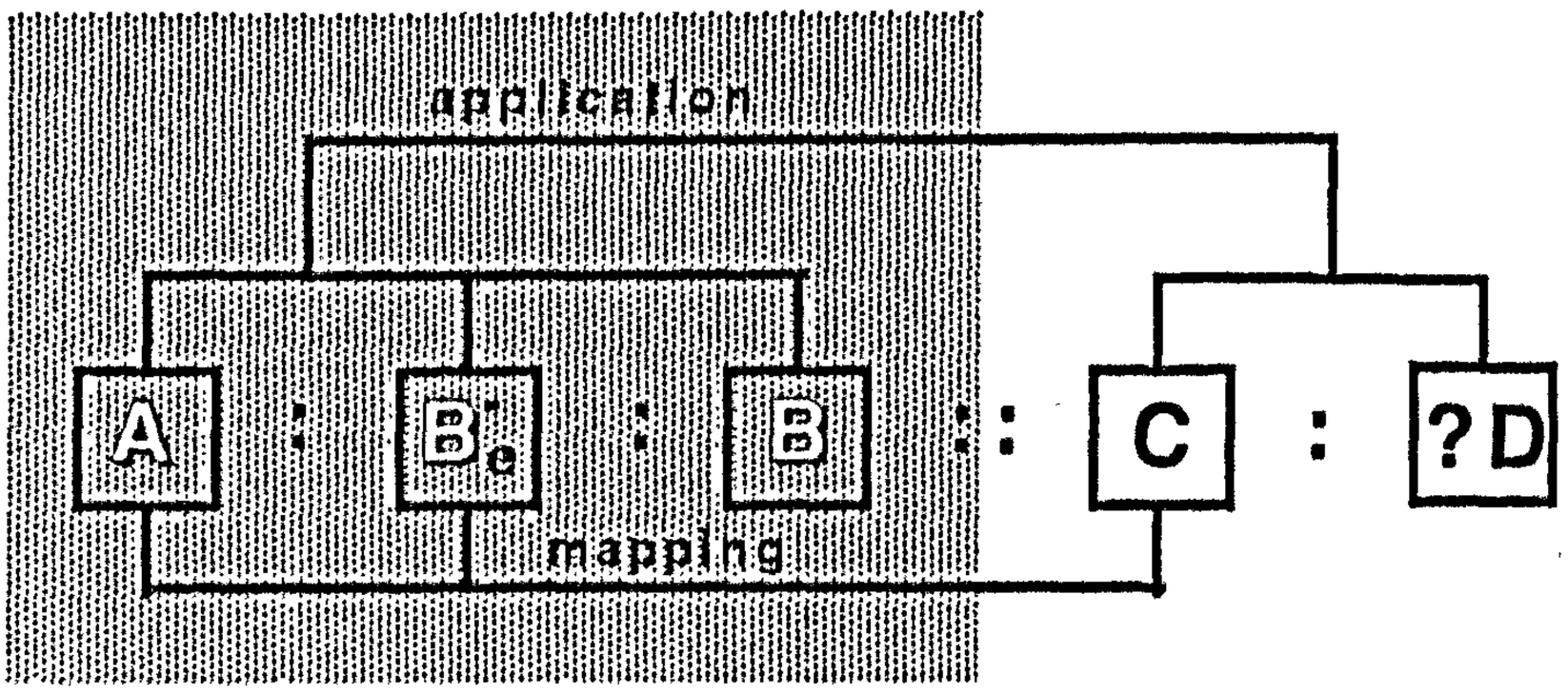


Figure 4.15. The solution of the example problem is no longer available for study.

There are three possible causes for failure to solve the problem which can be derived from figure 4.15. First, the solver may not remember the givens in the source, and hence may have difficulties finding correspondences between the relevant elements of the earlier problem and the target; second, the solver may not remember the relation between the givens in the source and the solution derived from them; third, the solver may not know how to apply the relation between the source A and B terms even if it is remembered since the higher-order relation is unknown, that is, information in the source problem may be lacking about how to transfer the solution from one problem to another.

4.2.2. Solving the related problem when the practice problem is available for study

Figure 4.16 shows the situation when the related practice problem is available for study. The givens in the source (the A term) should be mapped onto the target (C); the relation between the source A term and B term is explained (Be) and the whole problem and solution is available for study. It is the job of the solver to generalize the relation between A and B (the solution procedure) and apply it to the mapped elements in the target C term in order to generate the solution (D term). Nevertheless the solution depends on the adequacy of the mapping relations from source to target. In order to understand why more than half of the subjects still failed to generate a correct solution, a more detailed analysis is called for. This will be the subject of section 5.

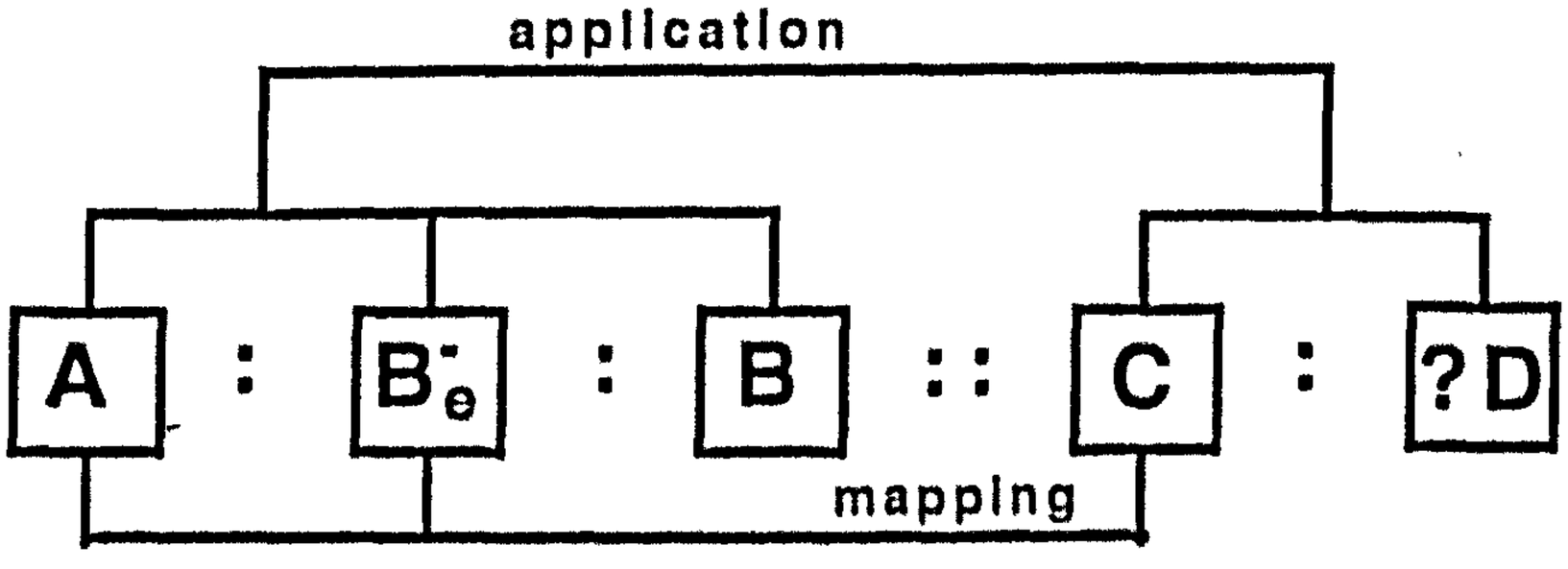


Figure 4.16. Solution to the previous problem is available to help solve current problem.

4.2.3. Blocked mappings from the unrelated practice problem

In all three experiments the unrelated practice problem[1] was not available for study when subjects were solving subsequent problems. The problem was unrelated to later problems in terms of its surface features as well as the underlying structure (the equation necessary to generate a solution). Subjects in all conditions were told that the earlier problem would help them solve later ones. This being the case, subjects would have tried to use the unrelated problem as a source, hence the access line in figure 4.17 from the C term to the A term, which they may take to be a relevant source problem. Even if the unrelated problem were available for study they would be unable to map the source problem givens onto the current problem (since they are unrelated) and to apply the relation between source problem statement and solution to the current problem. This is represented in figure 4.17 by the blocked lines.

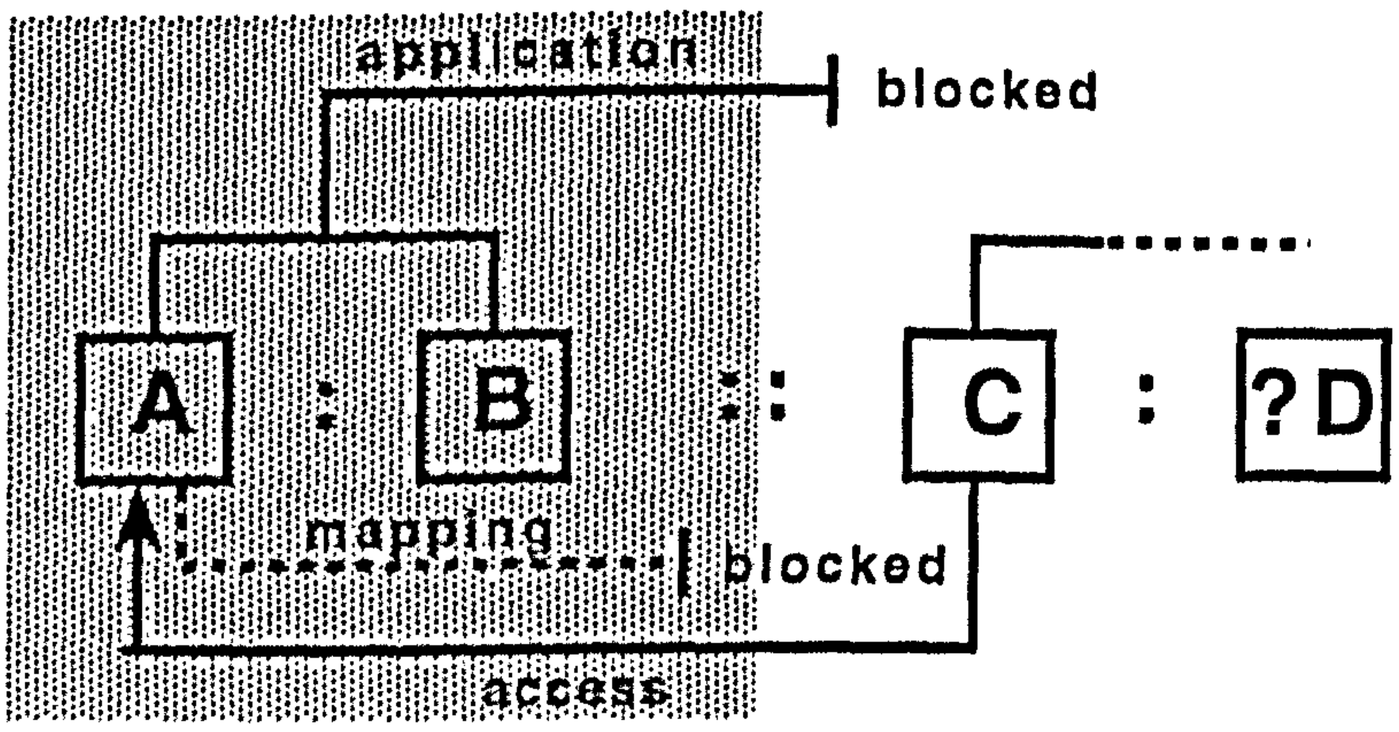


Figure 4.17. Unrelated practice problem

4.2.4. The effect of a better explanation of the mapping relations

The elaborated problem solution, represented by B_e in figure 4.18, is presumed to aid the mapping of the givens in the source to those in the target (the rates of the two vehicles and the times taken). It provides a method which includes constructing a table; this is an intermediate representation and is represented in figure 4.18 as B'. The success rate by students given the elaborated solution to study was far higher than for the unelaborated solution.

---

[1]*Although all practice problems presented in the experiments had an explanation of the solution these data were not provided for the unrelated practice problem. For that reason the problem is treated as if no explanation had been provided.*

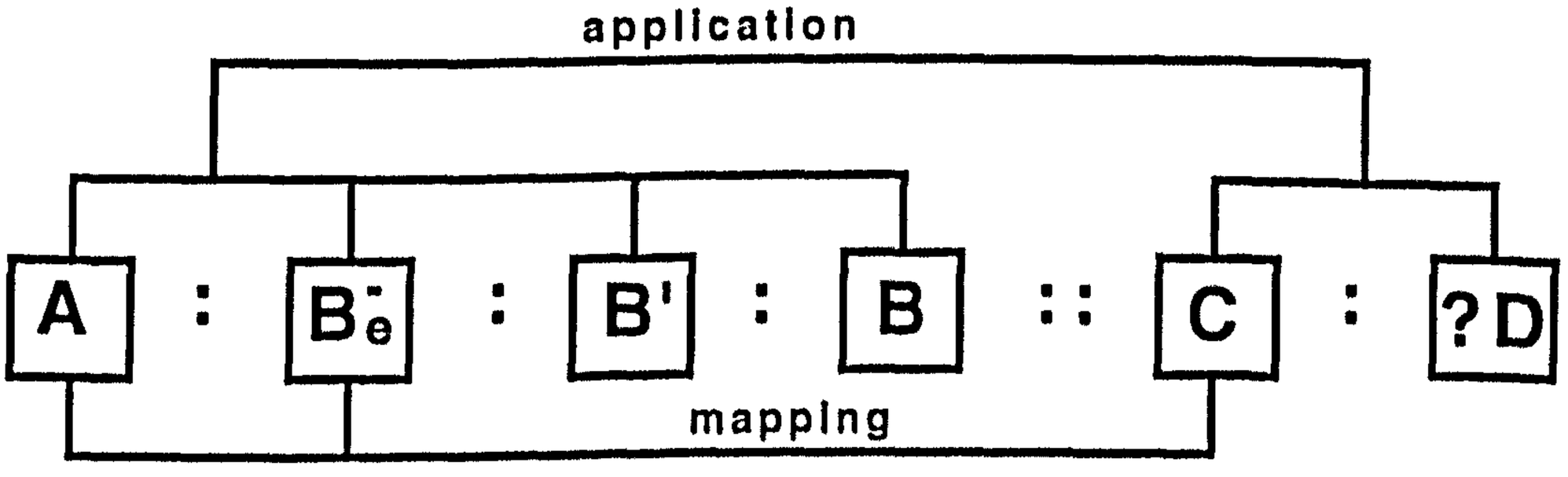


Figure 4.18. Previous elaborated solution is provided

4.2.5. Applying the elaborated solution from memory

When the elaborated solution is not available, as represented in figure 4.19, there was very little deterioration in results for problems which were equivalent (isomorphic). Constructing a table (the intermediate representation B' in the figure) has possibly allowed students to reconstruct more of the solution plan from memory, since the table provides a means of generating the correct equation, and of showing how the values in the problem can be inserted into it. That is, the elaborated explanation has possibly made it easier for the solver to make inferences about what subgoals are required and how the problem's givens and these subgoals relate to the relevant equation.

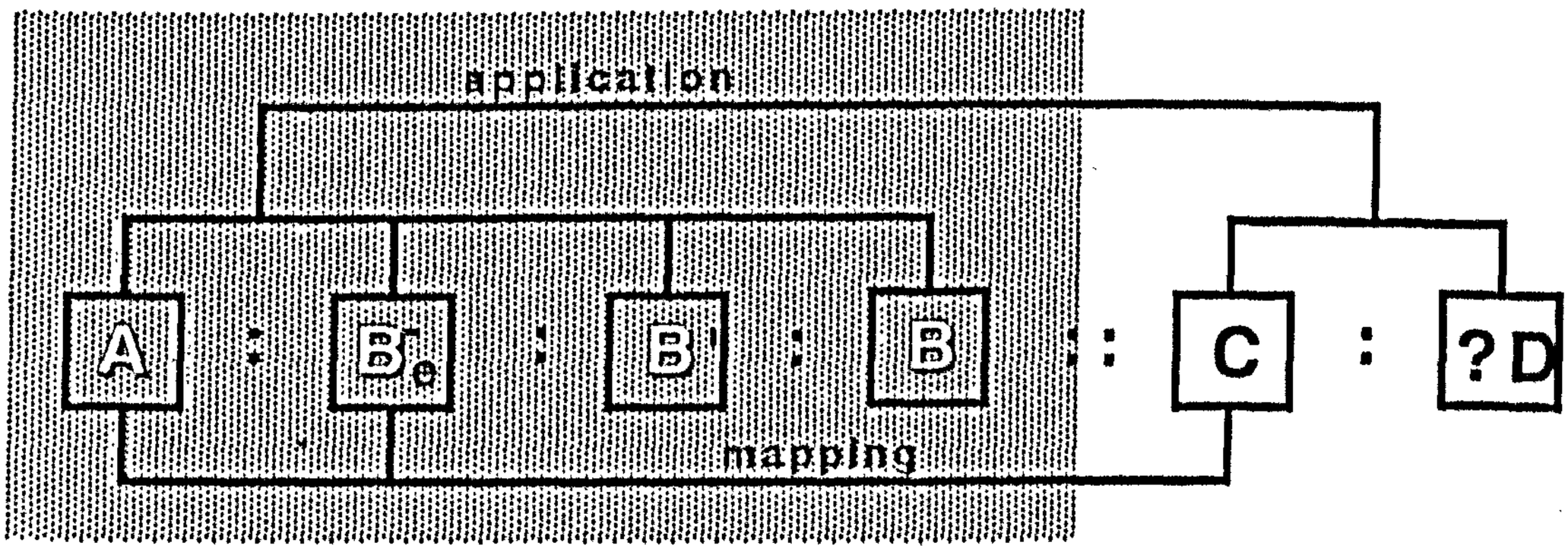


Figure 4.19. Previous solution is not available but the elaborated solution helps subjects to conceptualize the target problem.

4.3. Gaps in the explanations of the colinear algebra word problems

In the next two subsections I will examine where there are gaps in the explanations provided by Reed et al. in the practice problem and how this affects the students' approach to solving the close and distant variants.

4.3.1. Gaps in the explanation of the unelaborated practice distance problem.

Here is the related practice problem given to the students along with the explanation of the solution:

> *A car travelling at a speed of 30 miles per hour (mph) left a certain place at 10.00 a.m. At 11.30 a.m. another car departed from the same place at 40 mph and travelled the same route. In how many hours will the second car overtake the first car?*

The problem is a distance-rate-time problem in which

$$Distance = Rate \times Time$$

Because both cars travel the same distance, the distance of the first car (D_1) equals the distance of the second car (D_2). Therefore:

$$D_1 = D_2 \quad or \quad R_1 \times T_1 = R_2 \times T_2$$

where $R_1 = 30$ mph, $R_2 = 40$ mph, and $T_1 = T_2 + {}^3/_2$ hr. Substituting gives the following:

$$\begin{aligned} 30 \times (T_2 + {}^3/_2) &= 40 \times T_2 \\ 30T_2 + 45 &= 40T_2 \\ T_2 &= 4.5 \text{ hr.} \end{aligned}$$

4.3.1.1. The problem givens

Figure 4.20 shows the problem givens necessary for the solution. Each of the givens highlighted in the problem statement is represented by A^1, A^2, etc. in the order given in the problem. The goal (A^6_g) is also included as one of the problem givens. The reason for this will become clear later.

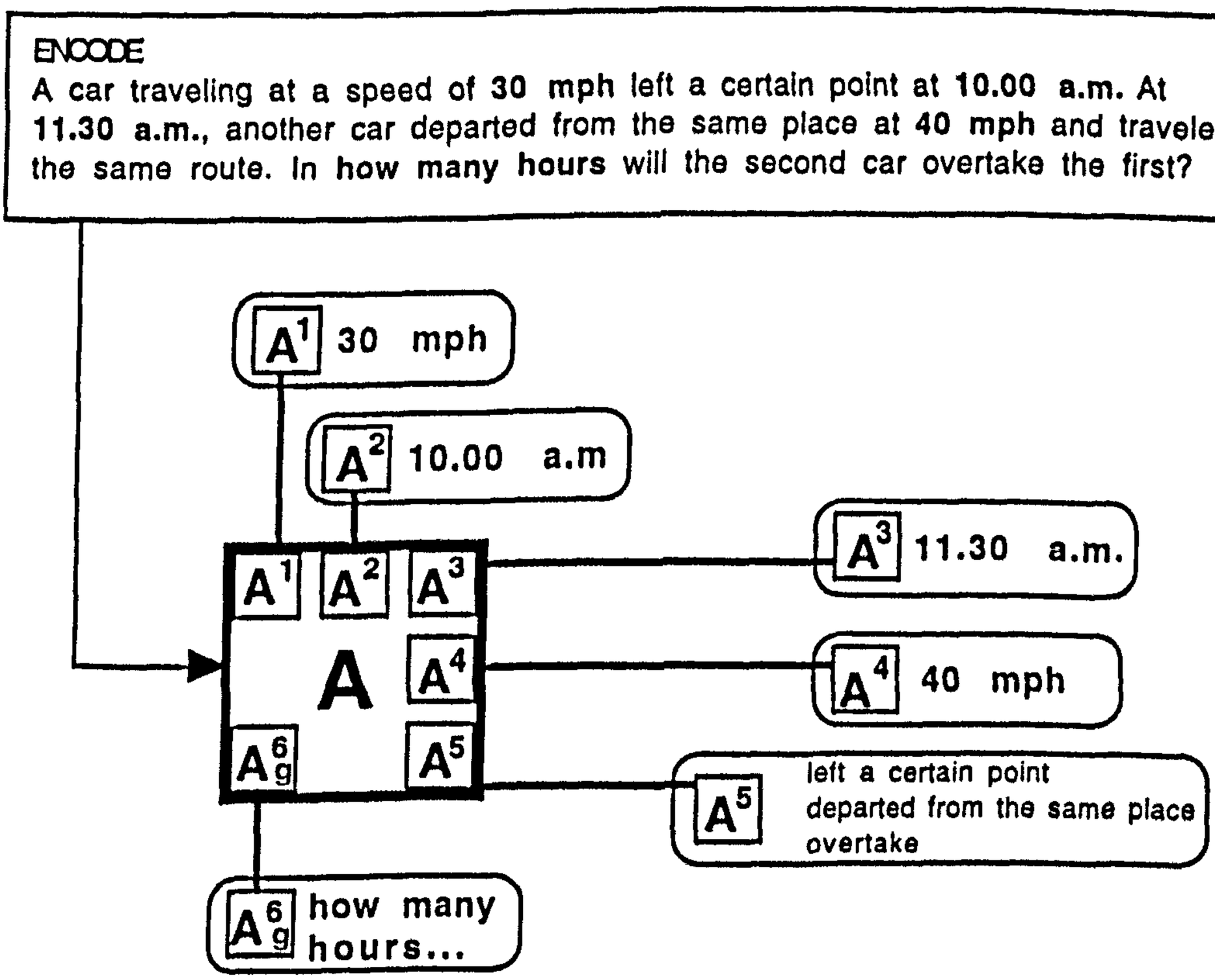


Figure 4.20. The practice 'distance' problem givens.

4.3.1.2. Representing the steps in the problem

The practice problem explains the solution procedure in a number of steps. Since it is meant as a model for other problems of the same type, it should contain features that allow the solver to generalize to other problems. Each of the steps in the explanation will be discussed in turn, along with a commentary on what it fails to explain.

Step 1. Understanding the first statement.

The problem is a distance-rate-time problem in which

Distance = Rate x Time

The first step is to identify the problem type. The explanation refers to it as a 'distance-rate-time' problem (B_e^1 in row *a* of figure 4.21) in which *distance = rate x time* (B^1 in figure 4.21*a*). However, we are not told what features of the problem make this a distance-rate-time problem, and there is no explanation as to why *distance* equals *rate x time*.

Step 2. Understanding the second statement.

Because both cars travel the same distance, the distance of the first car (D_1) equals the distance of the second car (D_2).

There is no explicit statement of how we know both cars travel the same distance. From the givens in the problem (A^5 in figure 4.20), the reader has to infer this in order to understand the explanation represented as Be^2 (hence the dotted line from A^5 to Be^2 in row *b*). The second subgoal to be attained is to make the distance travelled by car1 equal to the distance travelled by car2, as shown in B^2.

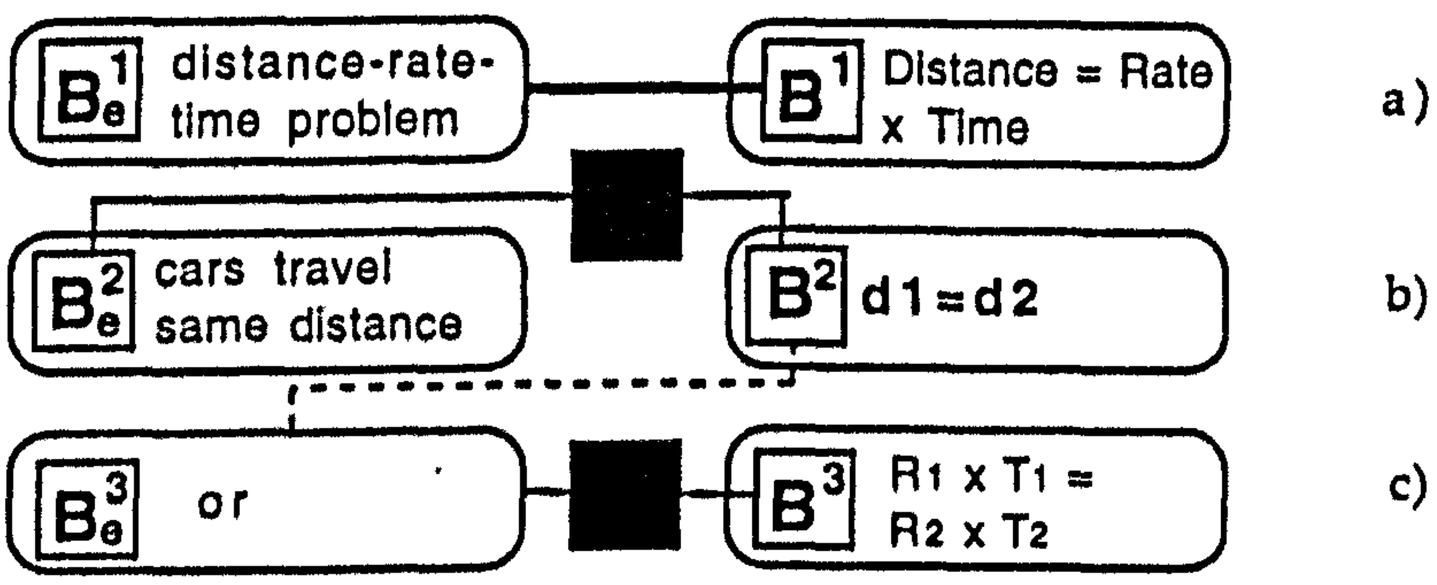


Figure 4.21. The first three subgoals of the practice distance problem

Step 3. Understanding equivalences mentioned.

Therefore: $D_1 = D_2$ *or* $R_1 x T_1 = R_2 x T_2$

a) There is no explanation of what T1 and T2 refer to.

b) There is no explanation of where $R_1 x T_1 = R_2 x T_2$ comes from;

c) Students may wonder what happened to the brackets round (D_1) and (D_2);

d) It is not clear whether the 'or' means that it does not matter which equation to use;

e) There is no explanation of why *Distance = Rate x Time* is no longer the appropriate equation;

The only explanation of the relation between D1 = D2 and R1 x T1 = R2 x T2 is the word 'or' (Be^3 in row *c* of figure 4.21). There is no explanation of why they are equivalent or why R1 x T1 = R2 x T2 is the relevant equation to use. The inferences involved here are:

If D1 = D2
and if D1 = R1 x T1
and D2 = R2 x T2
then R1 x T1 = R2 x T2

The word 'or' as an explanation inadequately captures the essence of this chain of inference. The black box in the figure in row *c* represents the fact that it is not made clear where B^3 comes from.

Step 4. Understanding terms used.

where $R_1 = 30\ mph$, $R_2 = 40\ mph$, *and* $T_1 = T_2 + {}^3/_2\ hr$

From the givens in the problem we know that car1 travels at 30mph and car2 at 40mph (A^1 and A^4 in figure 4.22 *d* and *e*). The explanation states that $R_1 = 30mph$ and $R_2 = 40mph$ (B_e^4 and B_e^5 respectively in rows *d* and *e*). The subgoals B^4 and B^5 involve mapping the givens, 30mph and 40mph, onto R_1 and R_2 in the equation. These are the only two mappings that can be made from the problem's givens directly onto the relevant equation.

The explanation states that $T_1 = T_2 + {}^3/_2$ (B_e^6 in figure 4.22). It does not state what T_1 and T_2 refer to (this is represented by the dotted inference lines from A^2 and A^3 in row *f*). It does not explain where the ${}^3/_2$ comes from, what operators should be applied to derive it, nor why T_1 should be equal to $T_2 + {}^3/_2$. This information is derived from the departure times of the two cars represented as A_2 and A^3 respectively. The three black boxes represent the missing information in row *f*).

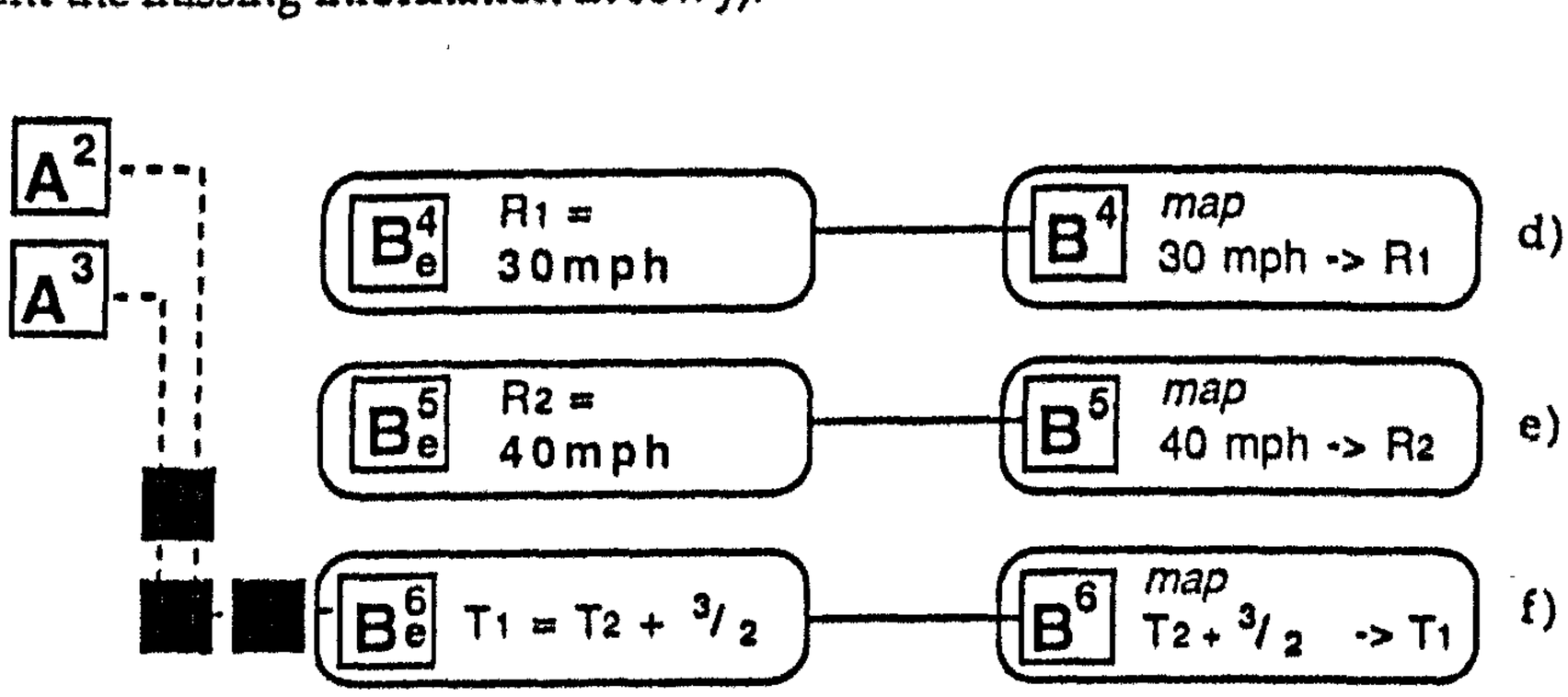


Figure 4.22. Subgoals 3 to 5 in the practice distance problem.

Step 5. Understanding 'Substituting.'

Substituting gives the following:

a) No explanation is given as to what should be substituted for what;

b) Nor is there a statement about which of the 2 equations (*Distance = Rate x Time* or $R_1 \times T_1 = R_2 \times T_2$) should be used.

The goal (A^6_g) is to find how many hours it takes the second car to overtake the first but there is no mention of how this relates to the result of the algebraic manipulations. An operator - substitution - is mentioned but there is no indication of what it operates on. This is the reason for the dotted line from B_e^7 to B^7 in row *g*. Nor are we told what has become of the units (*mph* and *hr*). Included in subgoal B^7 are the results derived from previous operations: B^4 and B^5 where the rates were mapped onto the equation; and B^6 where $T_1 = T_2 + {}^3/_2$ is derived. In the summary figure 4.24, the arrows in rows *d*, *e* and *f* show that they should be mapped onto the equation B^3 in row *c*.

The remaining steps involve the arithmetic manipulations of the terms of the equation (rows *g* to *i* in figure 4.23).

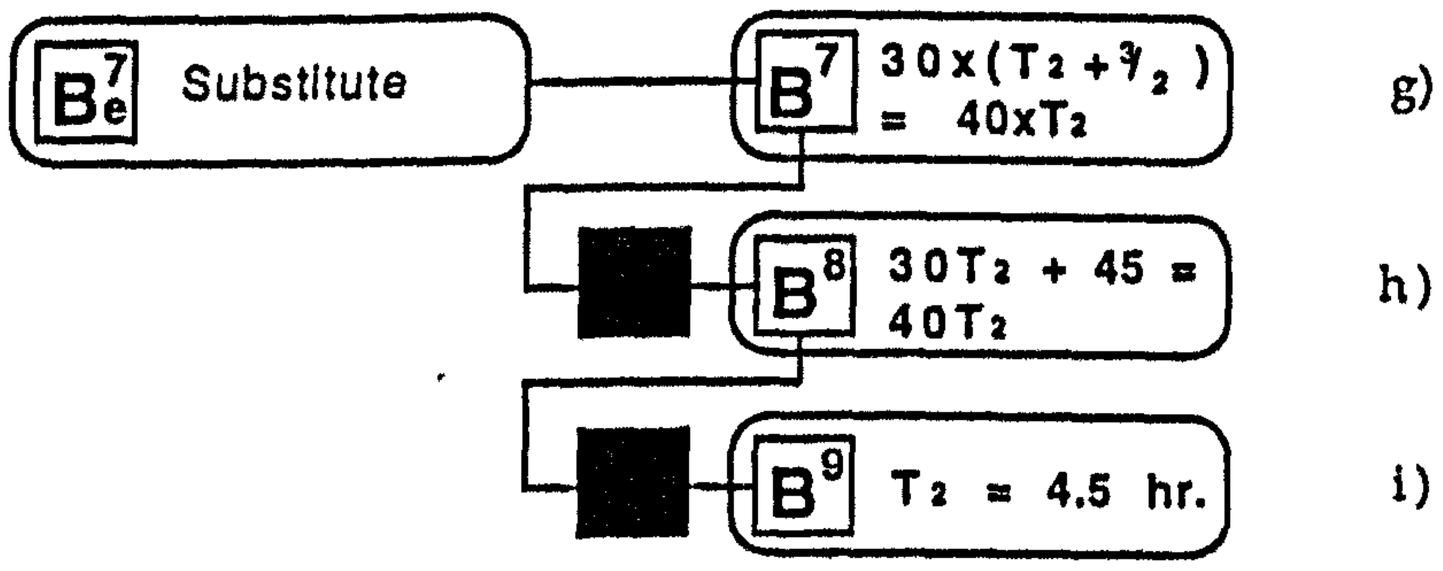


Figure 4.23. The final steps in the solution to the practice distance problem.

Step 6. Understanding first algebraic transformation.

$$30 \times (T_2 + {}^3/_2) = 40 \times T_2$$

a) There is no explanation of what has happened to 'mph' and 'hr';
b) There is no explanation of where the brackets have come from.

If we look at the transformations which have taken place from a purely naive point of view then solvers with little or no knowledge of algebraic manipulation will have seen the following transformations:

Transformation -> remove *mph*
Transformation -> remove *hr*
Transformation -> put brackets round last two terms on left-hand side

There is no indication of the operators used to effect these transformations.

Step 7. Understanding second algebraic transformation.

$$30 \times (T_2 + {}^3/_2) = 40 \times T_2$$
$$30T_2 + 45 = 40T_2$$

Transformation -> Remove multiplication signs

Transformation -> Don't change + signs

Transformation -> Remove brackets

Transformation -> Change $^3/_2$ to 45

Step 8. Understanding third algebraic transformation.

$$30T_2 + 45 = 40T_2$$
$$T_2 = 4.5\ hr$$

Transformation -> remove *30* from T_2

Transformation -> remove + sign from left-hand side

Transformation -> remove *45* from left-hand side

Transformation -> change $40T_2$ *to 4.5 hr*

In this final transformation, a large number of changes have taken place, again with no indication of the operators that have been applied to get from the first line to the second. The lack of explicit explanations in the last few steps gives rise to the black boxes in rows *h* and *i*.

Figure 4.24 presents a summary of the foregoing analysis. From the figure we can predict that solvers using this example will have difficulties wherever there are black boxes and dotted inference lines.

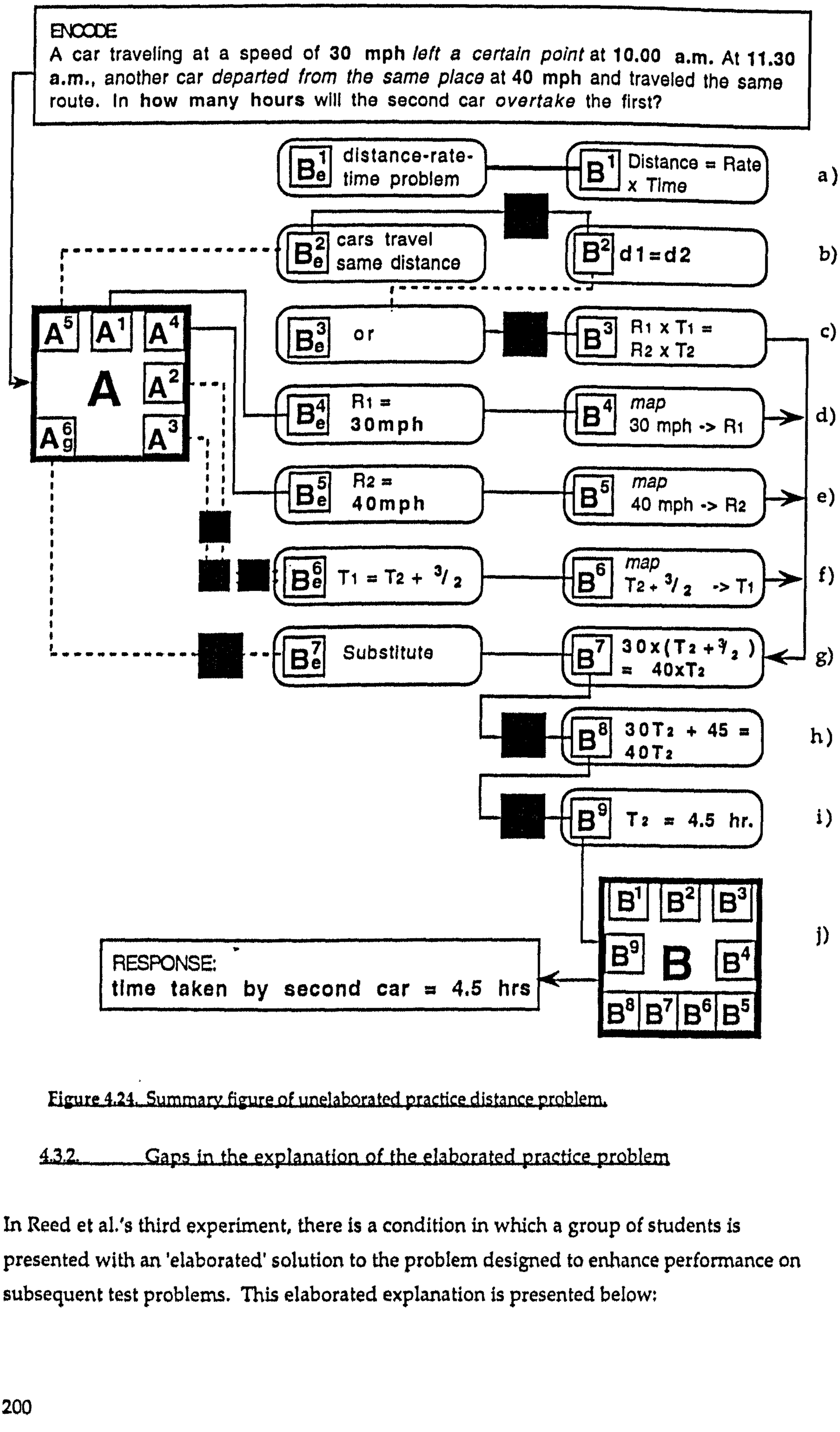


Figure 4.24. Summary figure of unelaborated practice distance problem.

4.3.2. Gaps in the explanation of the elaborated practice problem

In Reed et al.'s third experiment, there is a condition in which a group of students is presented with an 'elaborated' solution to the problem designed to enhance performance on subsequent test problems. This elaborated explanation is presented below:

A car travelling at a speed of 30 miles per hour (mph) left a certain place at 10.00 a.m. At 11.30 a.m. another car departed from the same place at 40 mph and travelled the same route. In how many hours will the second car overtake the first car?

The problem is a distance-rate-time problem in which

Distance = Rate x Time

We begin by constructing a table to represent the distance, rate, and time for each of the two cars. We want to find how long the second car travels before it overtakes the first car. We let t represent the number that we want to find and enter it into the table. The first car then travels $t + {}^3/_2$ hr *because it left* $1^1/_2$ *hrs earlier. The rates are 30 mph for the first car and 40 mph for the second car. Notice that the first car must travel at a slower rate if the second car overtakes it. We can now represent the distance each car travels by multiplying the rate and the time for each car. These values are shown in he following table.*

Car	Distance (miles)	Rate (mph)	Time (hr)
First	$30(t+{}^3/_2)$	30	$t+{}^3/_2$
Second	40xt	40	t

Because both cars have travelled the same distance when the second car overtakes the first, we set the two distances equal to each other:

$$30(t + {}^3/_2) = 40t$$

Solving for t yields the following:

$$\begin{aligned} 30t + 45 &= 40t; \\ 45t &= 10t; \\ t &= 4.5 \text{ hr.} \end{aligned}$$

Line 1. Once again the reader has to infer why this problem is a 'distance-rate-time' problem. In row *a* of in figure 4.25 there is a statement of the type of problem this is ($B_e{}^1$ in row *a*). $B_e{}^2$ in row *b* requires the solver to construct a table.

Line 5. The goal of the problem is restated in the explanation: 'we want to find out how long the second car travels'. The explanation states that time should be represented by the variable, *t*. That is, the goal is stated in terms of a variable whose value has to be found by algebraic manipulations (row *c* of figure 4.22).

Line 6. The time taken by the first car is in turn related to *t* since it left a known number of hours earlier than *t*. This is the explanation of where the $t+{}^3/_2$ comes from, but we are

left to infer the origin of $^3/_2$. It does not explain that it comes from the times of the vehicles or what you have to do to derive the $1^1/_2$. This is represented by the dotted lines and black box between A^2 and A^3 in row *d*)

Line 9. There is no explanation of why the distance can be represented by multiplying the rate by the time for each car since the reader is not referred back to the distance equation, *nor is the relevant equation given.* For this reason $\mathbb{B}^7$ appears as a ghost term since it is not explicitly stated. The fact that both cars travel the same distance is not mentioned in the problem statement. These inferences are shown in row *g*.

Line 11. The table. Distance, rate and time can then be added to the table thereby providing all the values that can be fitted into the distance equation. However, the equation which appears most salient is the one provided in line 2, i.e. *Distance = Rate x Time*. There are 6 values represented in the table and the student may have difficulties mapping them onto this equation particularly as the equation is not the relevant one. The relevant equation is $Rate_{car1} \times Time_{car1} = Rate_{car2} \times Time_{car2}$ but *it is not provided anywhere in the elaborated explanation.* The black boxes between $\mathbb{B}^7$ and B'^8, and B^9 in figure 4.25 indicate where this information is missing and hence where solvers are liable to experience difficulties.

Line 20. An extra step has been included in the solution to the algebra problem. The transformations involved here are different from those in the 'unelaborated' problem explanation.

$$30t + 45 = 40t;$$
$$45 = 10t;$$

Transformation -> removẽ 30t from left-hand side

Transformation -> change 40t to 10t

Transformation -> remove plus sign from left-hand side

Line 21. Final transformations.

$$45 = 10t$$
$$t = 4.5\ hr.$$

Transformation -> change 45 to t

Transformation -> change 10 to 4.5

Transformation -> change t to *hr.* on right-hand side

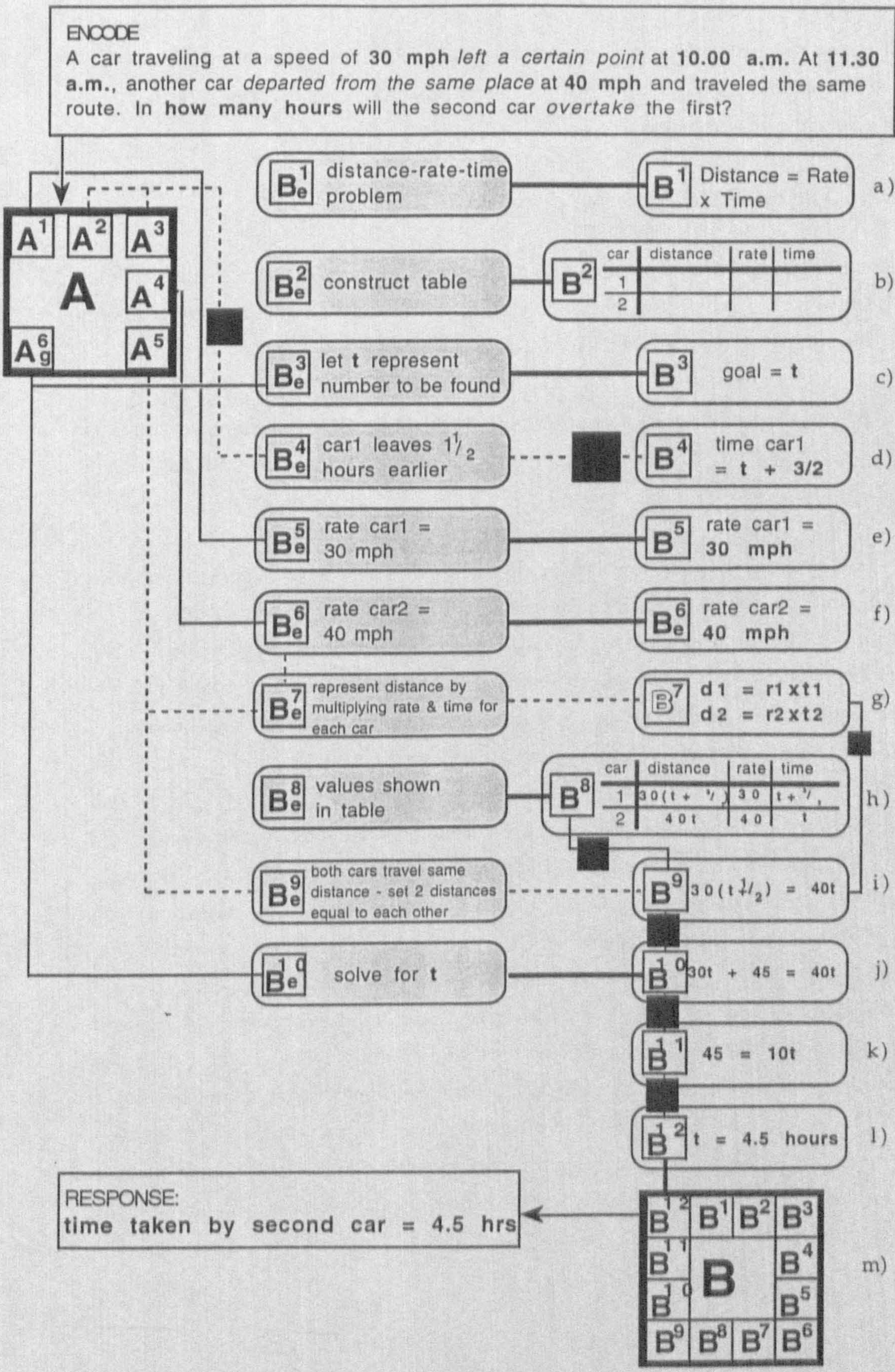


Figure 4.25. The elaborated practice distance problem.

There was a dramatic improvement in the solution rates in the condition in which an elaborated practice problem was presented. Compared to the explanation of the mappings in the unelaborated problem the solver can rely less on inferences. However, they have not been entirely eliminated.

4.4. The unrelated practice problem

Here is the practice problem which was presented prior to the distance problems. It is unrelated to the distance problems in that it relies on a different equation for its solution and the surface features are entirely different.

> *A baseball team's batting average of 0.263 is 0.004 greater than the average for the slugger and the pitcher. If the slugger's average is 0.192 greater than the pitcher's, find their averages.*

None of the surface features nor the underlying features have anything in common with the subsequent distance problems. The goal-subgoal structure of this problem is simpler than that of the practice distance problems and the test problems. This problem deals with averages but is not a *class* of problem as the distance problems are since it does not involve a particular equation which applies to a number of examples. The equation necessary for its solution is relevant only to this particular problem.

In order to solve the problem, the solver first has to generate the relevant equation (B^1 in row *a* of figure 4.26). This is not given in the problem statement and has to be inferred (hence the black box before subgoal B^1). Once the relevant equation has been generated then the values have to be discovered for the variables in the equation. First, the pitcher's average is assigned some variable, p in the figure below (subgoal B^2). If p is known then we know the average of the slugger (s) since it is 0.192 greater than p (subgoal B^3). The student has to know that adding these and dividing by 2 gives the average for the slugger and the pitcher (subgoal B^4) and adding 0.004 in turn to this result gives us the team average (subgoal B^5). Solving for p involves algebraic manipulations which the solver needs to know how to perform in order to find the solution. Solving for p generates the average of the pitcher (subgoal B^8) but the question also requires the average of the slugger which is the solution to subgoal B^9.

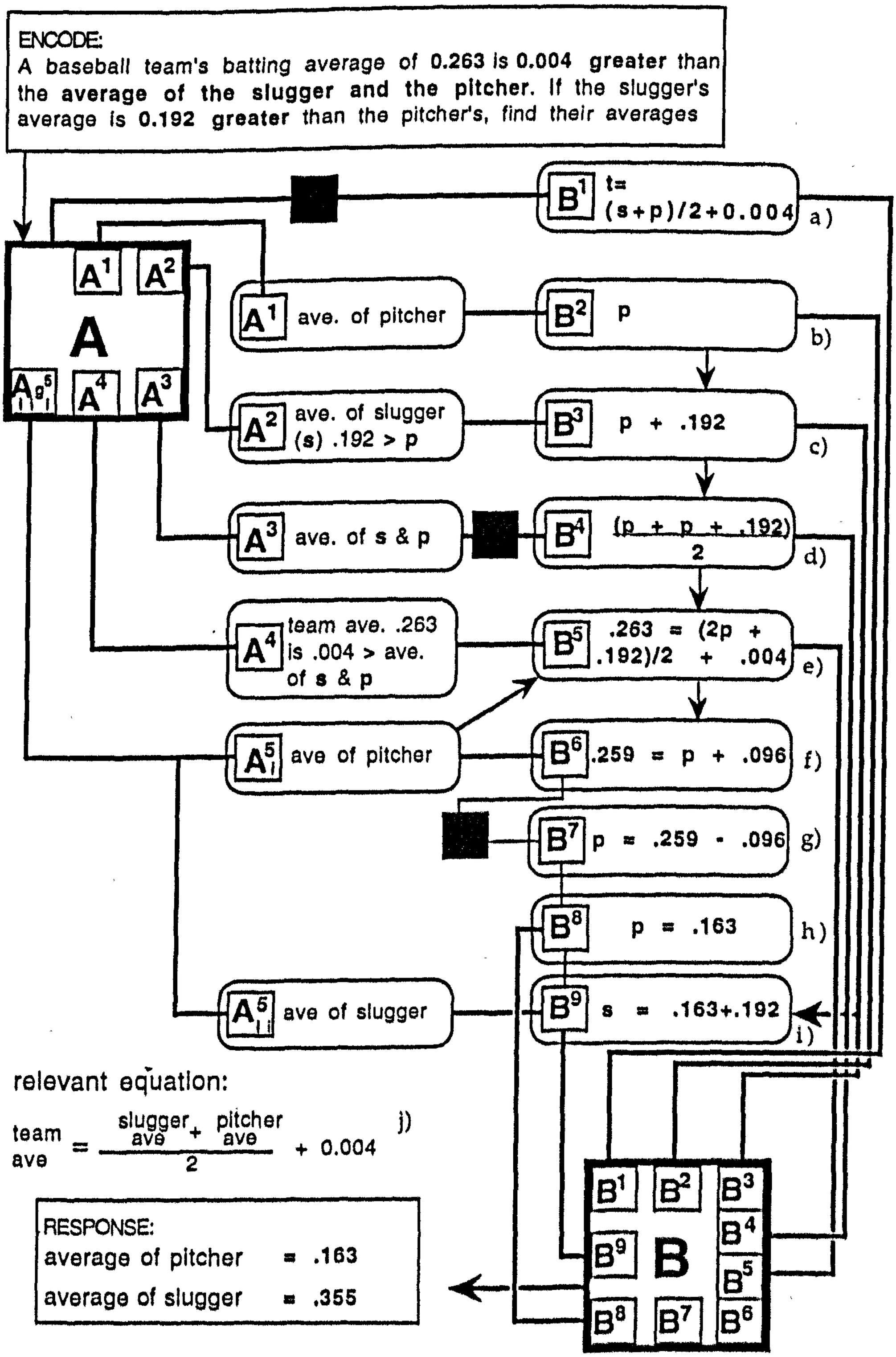


Figure 4.26. Summary figure of the unrelated practice problem: averages

5. The test problems

This section examines how the interpretation theory can be used to illustrate how well the example problems described above can be mapped onto the test problems Reed et al. gave their subjects. If the elaborated solution provides enough information about the mappings and relations between the example problem givens and solution, and onto the givens in an equivalent test problem, we should expect there to be very few inferences required on the part of the solver. Furthermore, if the elaborated solution provides enough information about the problem *type* then we would expect it to enable the solver to use it to solve a 'similar' problem of the same type.

5.1. Mapping the unelaborated training problem onto an equivalent test problem.

The following is the 'equivalent' distance problem from Reed et al.:

> *A car travels south at the rate of 30 mph. Two hours later, a second car leaves to overtake the first car, using the same route and going 45 mph. In how many hours will the second car overtake the first car?*

The practice problem states that the first car leaves 'from a certain place' and that the second car leaves 'from the same place.' This is not stated as explicitly in the test problem. The first car simply 'travels south' and a second car 'leaves to overtake the first car.' The problem givens, highlighted in bold typeface, do not match all of those in the training problem. There are 4 items in the source problem relevant to the solution, and only 3 in the test problem above. There is no need to calculate the difference in departure times in the test problem. The solution depends on generalizing *from* $T_1 = T_2 + 3/2$ *to* $T_1 = T_2 + d$, where d is the difference in departure times, and then mapping from the problem givens:

2 hours later, a second car leaves...

$T_1 = T_2 +$ d

The other elements that can be mapped onto the equation are:

a car travels at... 30mph. a second car... going 45mph

$R_1 \times T_1 =$ $R_2 \times T_2$

$T_1 = T_2 + 2h$

$R_1 \times T_1 = R_2 \times T_2$

We will look first at how the givens in this problem map onto the unelaborated practice problem. The problem givens are represented in figure 4.27.

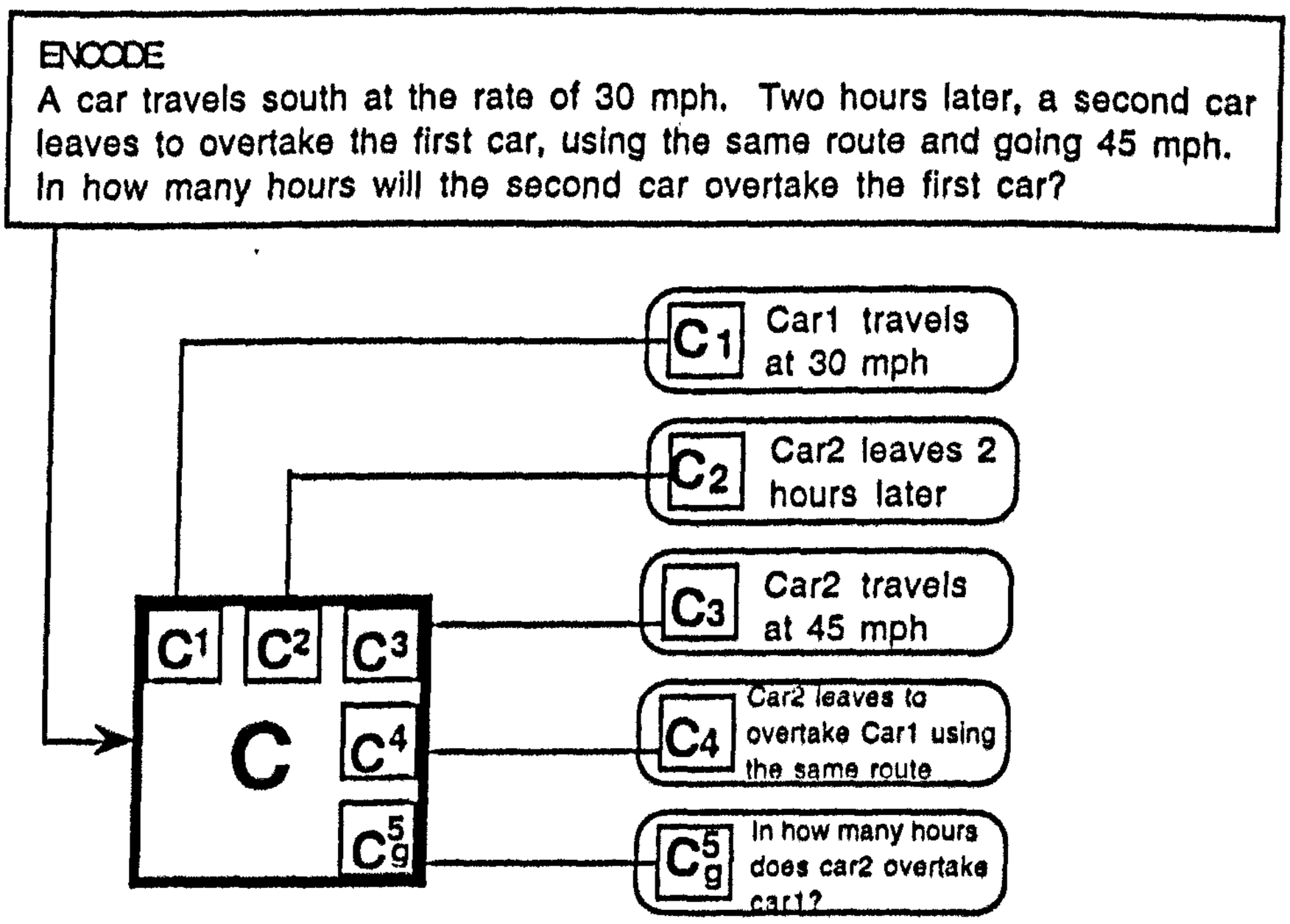


Figure 4.27. 'Equivalent' exercise problem givens.

The A and B terms in the figures that follow refer to the problem givens and subgoals in the practice distance problem analysed in figure 4.24.

Row *b* of figure 4.28 shows a dotted line from the A term to the B term and a black box indicating that the explanation in the original problem is incomplete (the rows are lettered in conformity to the summary figure 4.31).

In the test problem (represented by the C and D terms), not only do students have to infer that both cars travel the same distance, but also that they both leave from the same

point. In the previous explanation students are told in B_e^2 that the distance travelled by both cars is the same.

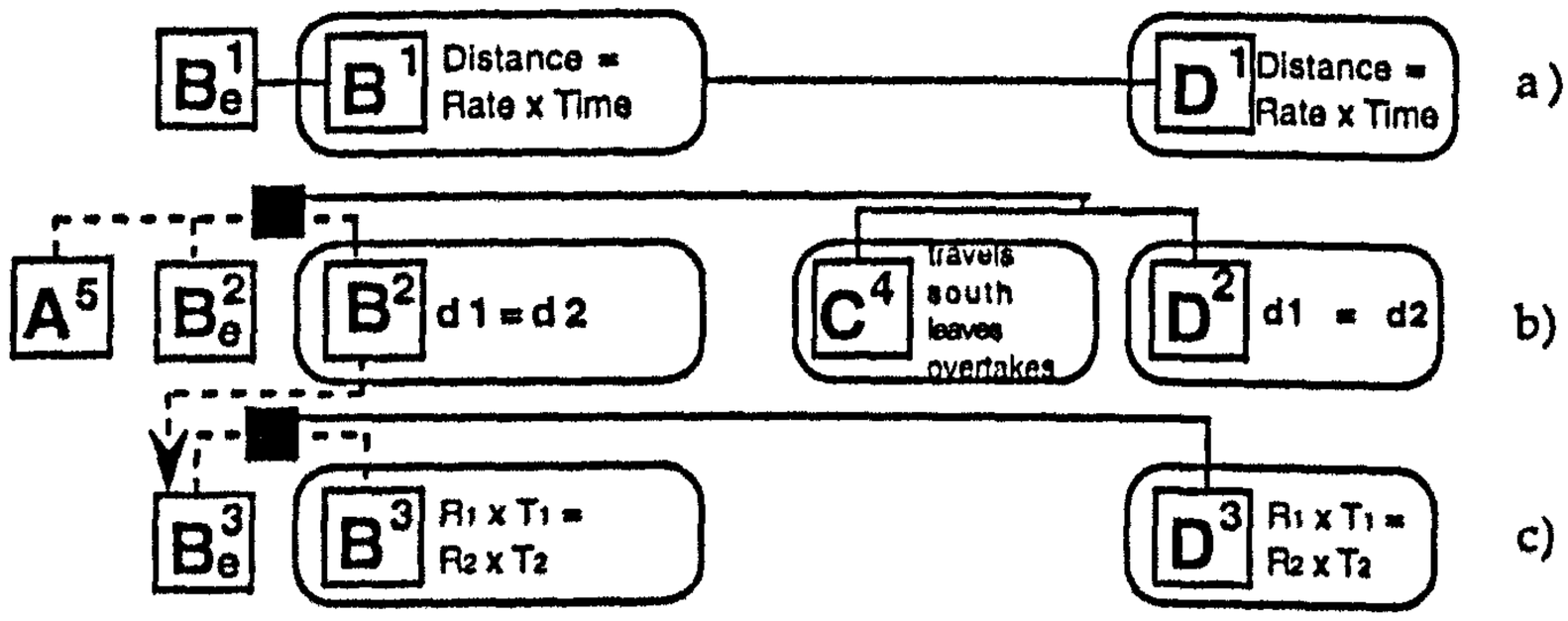


Figure 4.28. Inferences required in understanding equivalence of distance travelled in test problem.

Three of the problem givens can be slotted into the relevant equation. The first of these, 30mph, is identical to the first element of the training problem (shown as A^1 in figure 4.29, row *d*). B^4 is the subgoal which maps 30mph onto *R1* in the equation $R_1 \times T_1 = R_2 \times T_2$. This mapping can be imitated in the current problem to generate subgoal D^4.

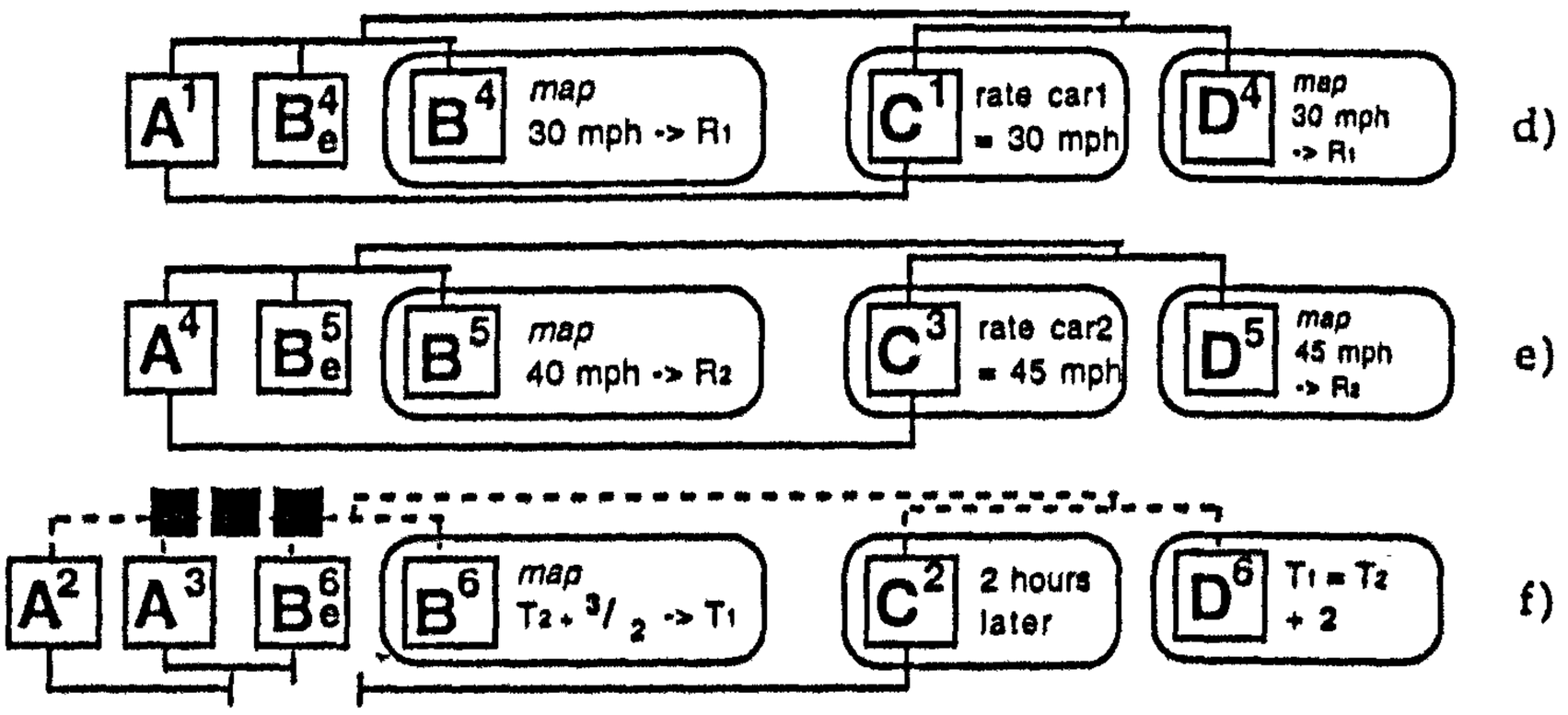


Figure 4.29. Mapping the first of the training problem givens and the second of the test problem givens.

Similarly, the rate of the second car can be mapped across as illustrated in row *e* of figure 4.29. Both rates can now replace the variables in the equation in D_3 in row *c* of figure 4.28.

In row *f* there is nothing in the earlier problem to map onto the current one. This mapping is therefore blocked (represented by the broken line between the C term and the A terms of the earlier problem). In the earlier problem a number of inferences are required to determine where the $^3/_2$ comes from (represented by the black boxes in row *f*). This figure is included in the subgoal given in B_e^6 .

The solver has to understand how to state the time taken by one car in terms of the time taken by the other in order to achieve this subgoal.

Finally, the goals in the practice and test problems are the same(A^6_g and C^5_g respectively in row *g* of figure 4.30). The difficulty here is that the earlier explanation does not include any mention of what the goal is in terms of the outcome of the algebraic manipulations. No explanation is given in the earlier problem about how to solve for T_2, hence the black boxes between the subgoals B^8 and B^9.

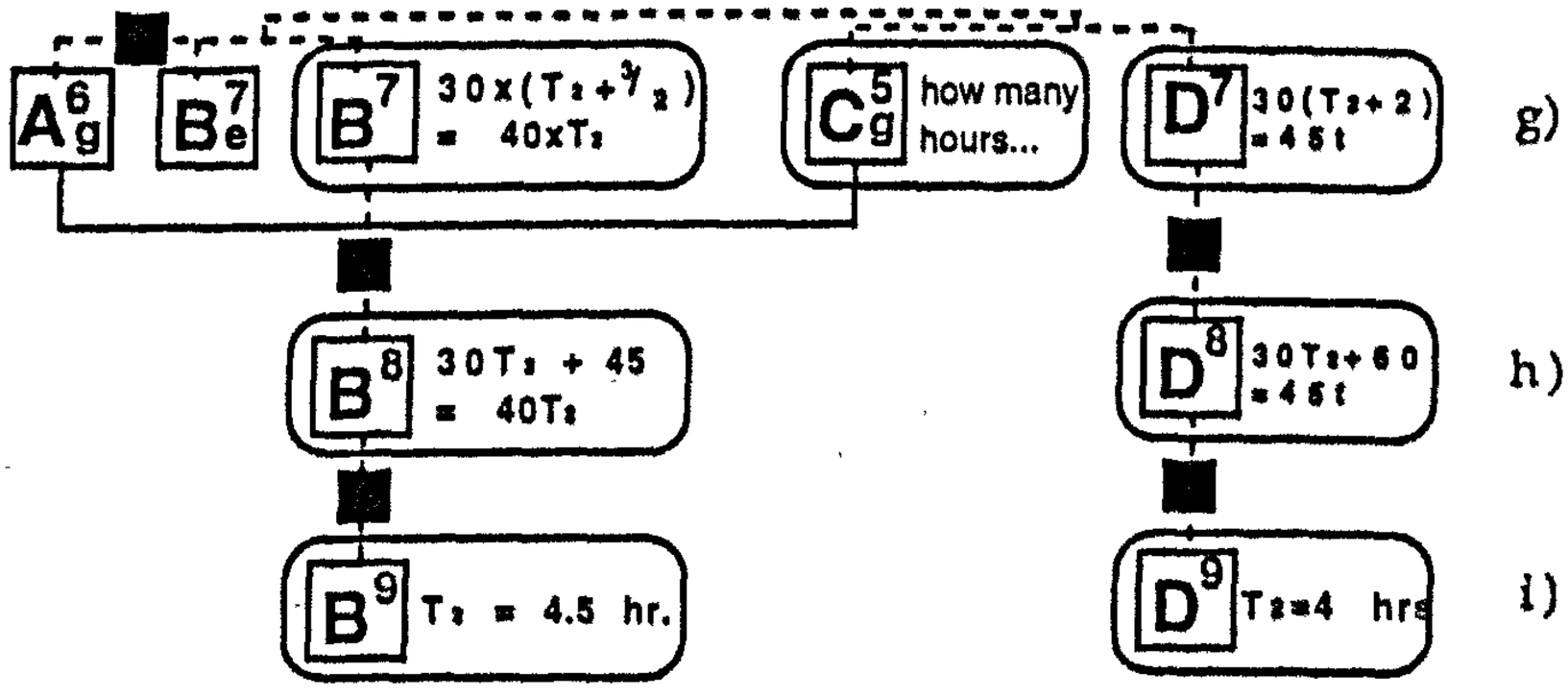


Figure 4.30. Inference required in adapting goal of training problem to test problem.

Figure 4.31 summarizes the above showing how the problem givens are related to the solution when the previous example is accessed.

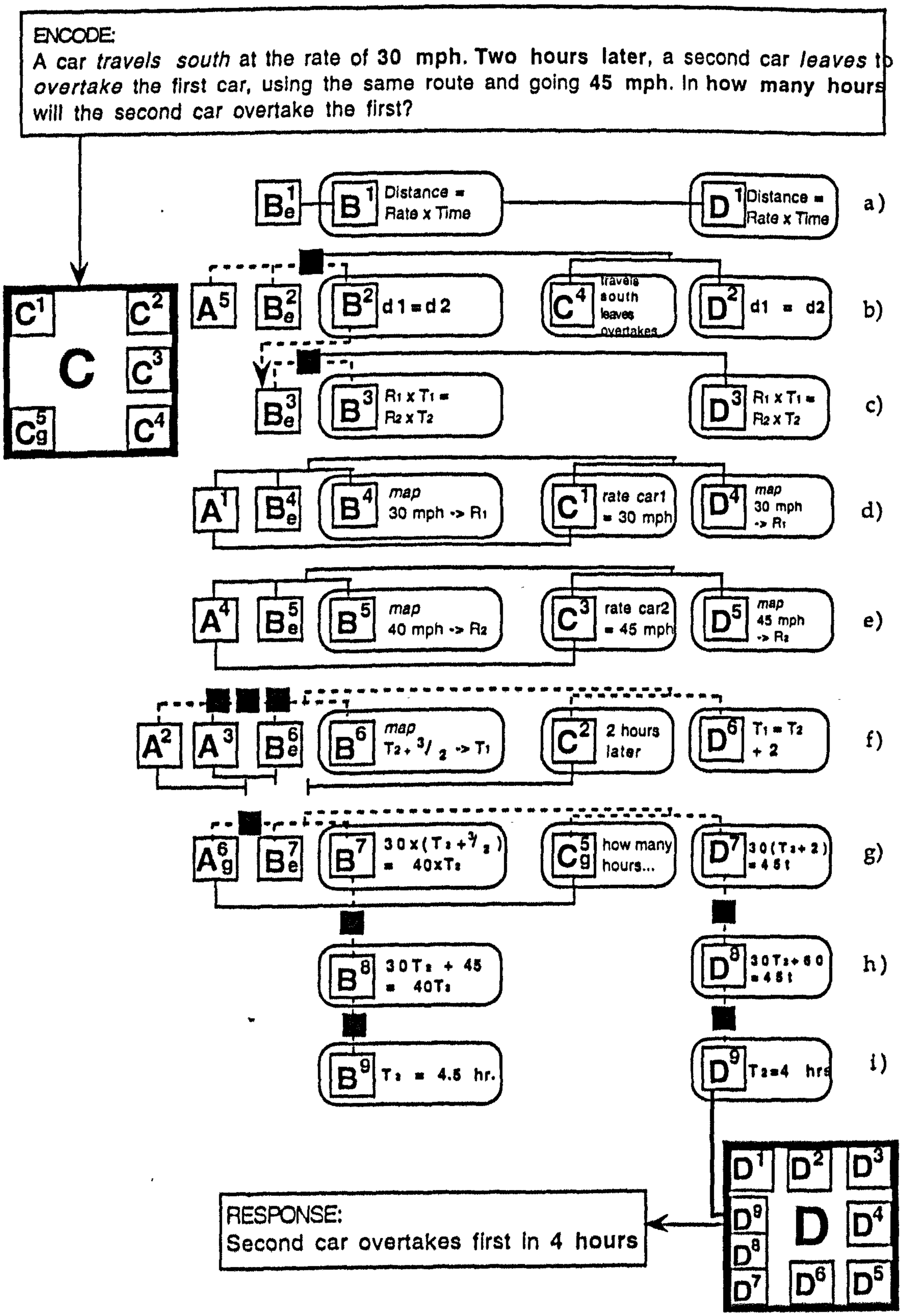


Figure 4.31. Analogical mapping of equivalent distance problem onto previous (unelaborated) example. Summary figure.

5.2. Mapping the elaborated training problem onto an equivalent test problem

In the diagrams that follow, the A and B terms represent the earlier elaborated distance problem analysed in figure 4.29. Looking back to that solution explanation, the subject would find that the first three subgoals are:

a) D^1 - classify the problem as a *distance = rate x time* problem. This was explained in subgoal B^1 of the earlier problem; so $B^1 \Rightarrow D^1$ (where '$\Rightarrow$' means 'is mapped to).

b) D^2 - construct a table ($B^2 \Rightarrow D^2$) and,

c) D^3 - generate a variable name for the goal ($B3 \Rightarrow D3$). These correspond to lines *a*, *b*, and *c* in figure 4.32.

Some of the elaborated training problem givens can be mapped directly onto the equivalent test problem (figure 4.32 lines *e* and *f*). The first car in both problems travels at 30mph and this can be mapped onto the relevant equation (assuming the equation can be generated - this is a separate subgoal of the problem). By changing the rate of the second car in the training problem from 40mph to 45mph in the test problem the solver could also map this onto the rate term in the equation. (The lettering of the rows in the following figures correspond to the lettering in the summary figure 4.34.)

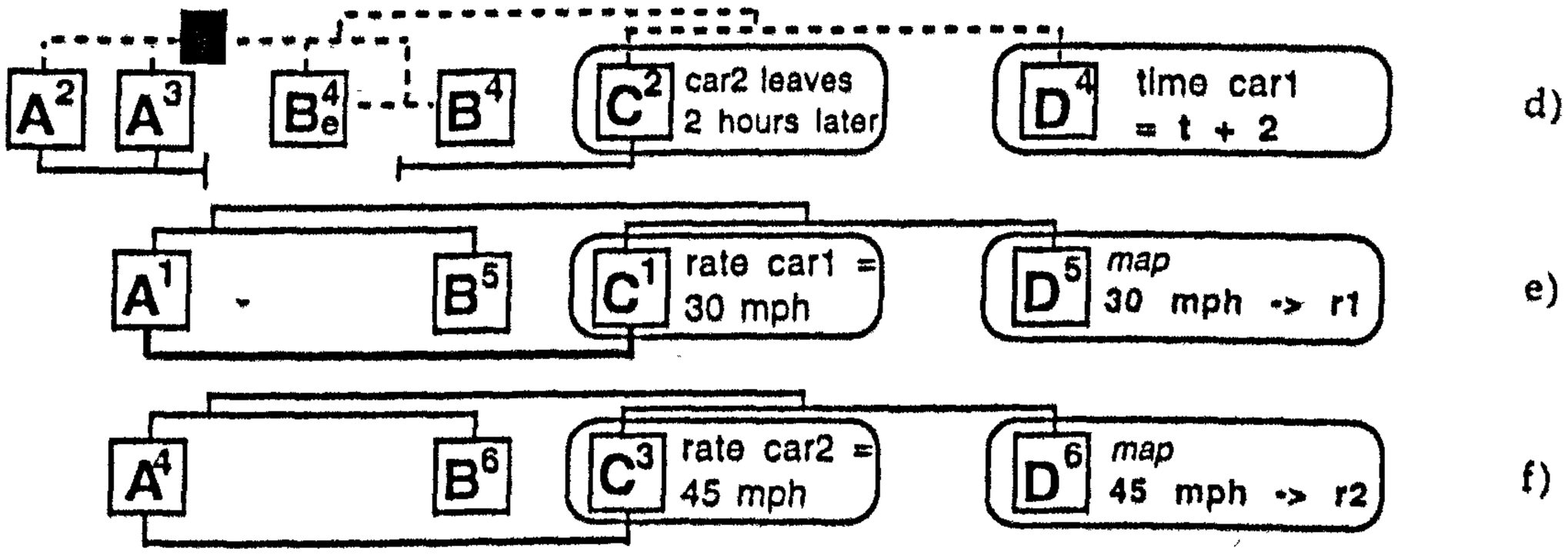


Figure 4.34. Mapping problem givens in test and elaborated problems.

The goals, represented by C^5_g in the test problem and A^6_g in the training problem, both ask for the *time* taken by the second car to catch up with the first one. In the training problem subjects are told to represent time by t. However, as figure 4.34*d* shows, there is no direct match between the '2 hours' mentioned in the test problem (C^2 in the figure) and the times given in the training problem (A^2 and A^3). The mappings between the two problems are therefore shown as being blocked. An inference is required to relate the times in the first

problem to the difference in times in the second. D^4 represents the fact that the time taken by car1 equals the time taken by car2 plus 2 hours.

The solver must infer that both cars travel the same distance. The givens in the problem which allow this inference is shown as C^4 in figure 4.35 row *g* ('leaves to overtake using same route'). This information is similar (but not identical) to that given in the training problem shown as A^5. Since the descriptions are not identical, A^5 and C^4 are linked by a dotted line. Subjects have to understand the explanation in B^7e that distance should be represented in terms of *rate x time* for both cars in order to generate the equation given in D^7. The inferences that were necessary to generate $\mathbb{B}^7$ are represented by the dotted lines between A^5, B^7e and $\mathbb{B}^7$. Those inferences have to be made in order to generate D^7. The subgoals included in D^7 represent the information derived both from the problem statement and from the previous subgoals. These can now be mapped onto the equations. The difficulty is that the equation is not explicitly stated in $\mathbb{B}^7$.

The table (shown in figure 4.35*h*) contains six boxes, representing the distance, rate and time for each car. The relevant equation has only four variables *rate x time* for each car. Students may find difficulty in relating the distance boxes to the rate and time boxes since all they have to go on is the statement in box B^7e 'represent distance by multiplying rate and time for each car.' Furthermore the relevant equation is *rate1 x time1 = rate2 x time2*. The only equation given in the example is *distance = rate x time*.

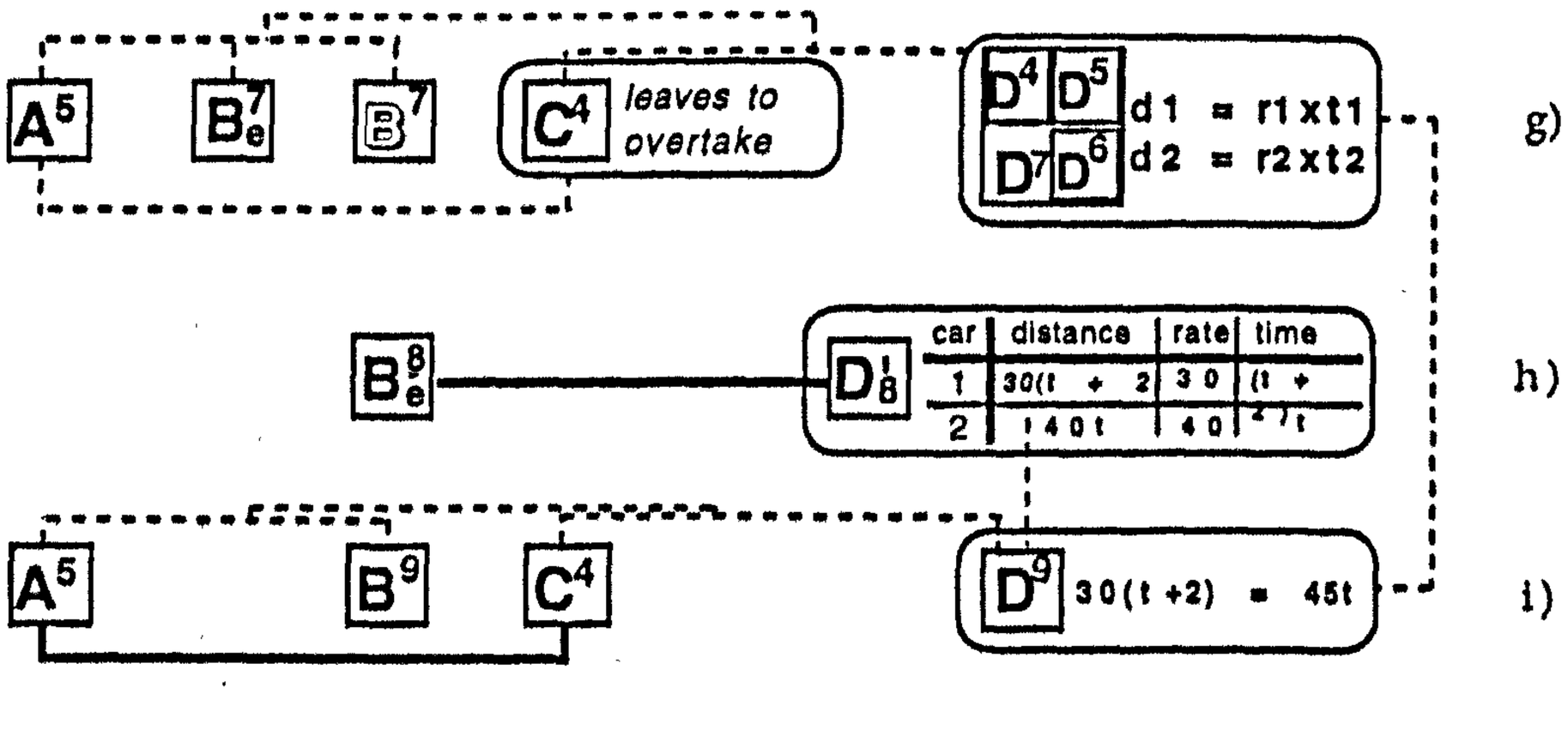


Figure 4.35. Mapping the relevant elements of the problem onto the equation and the table

The above analysis is summarized in figure 4.36. For clarity the processes involved in each row are briefly repeated below.

Row:

a) The first step is to recognize the problem as an example of the same type of problem as B^1 - Distance = Speed x Time.

b) Subjects are asked to construct a table as an intermediate representation ($B'^2 \Rightarrow D'^2$).

c) The goal is the same as in the training problem and, according to the explanation in B^3e, should be represented as t.

d) This subgoal is to generate the time taken by car1. This involves a number of inferences because the problem givens do not map onto the earlier problem. Further inferences are required to understand the training problem in the first place.

e) The rate of car1 in the training problem is identical to the rate in the test problem ($B^5 \Rightarrow D^5$).

f) The rate of car2 in the training problem maps directly to the rate in the test problem ($B^5 \Rightarrow D^5$).

g) The next subgoal is to represent distance by multiplying rate times time as explained in B^7e. No equation is explicitly given in the earlier problem for the subject to map to the current one.

h) This subgoal involves filling in the boxes in the table with the values from the problem statement and those derived from earlier subgoals.

i) The equation can now be generated from elements in the table. This depends on the subject understanding the relationships between the elements in the table and the explanation in row *g*. The subject has to understand that the distances travelled by both cars are the same. This information has to be inferred from A^5 and C^4

j, k, l) These are the next stages in solving for t. No explanation is given about what operators to apply to get from one stage to the next.

m) This represents the final solution with the subgoals 'nested' within it.

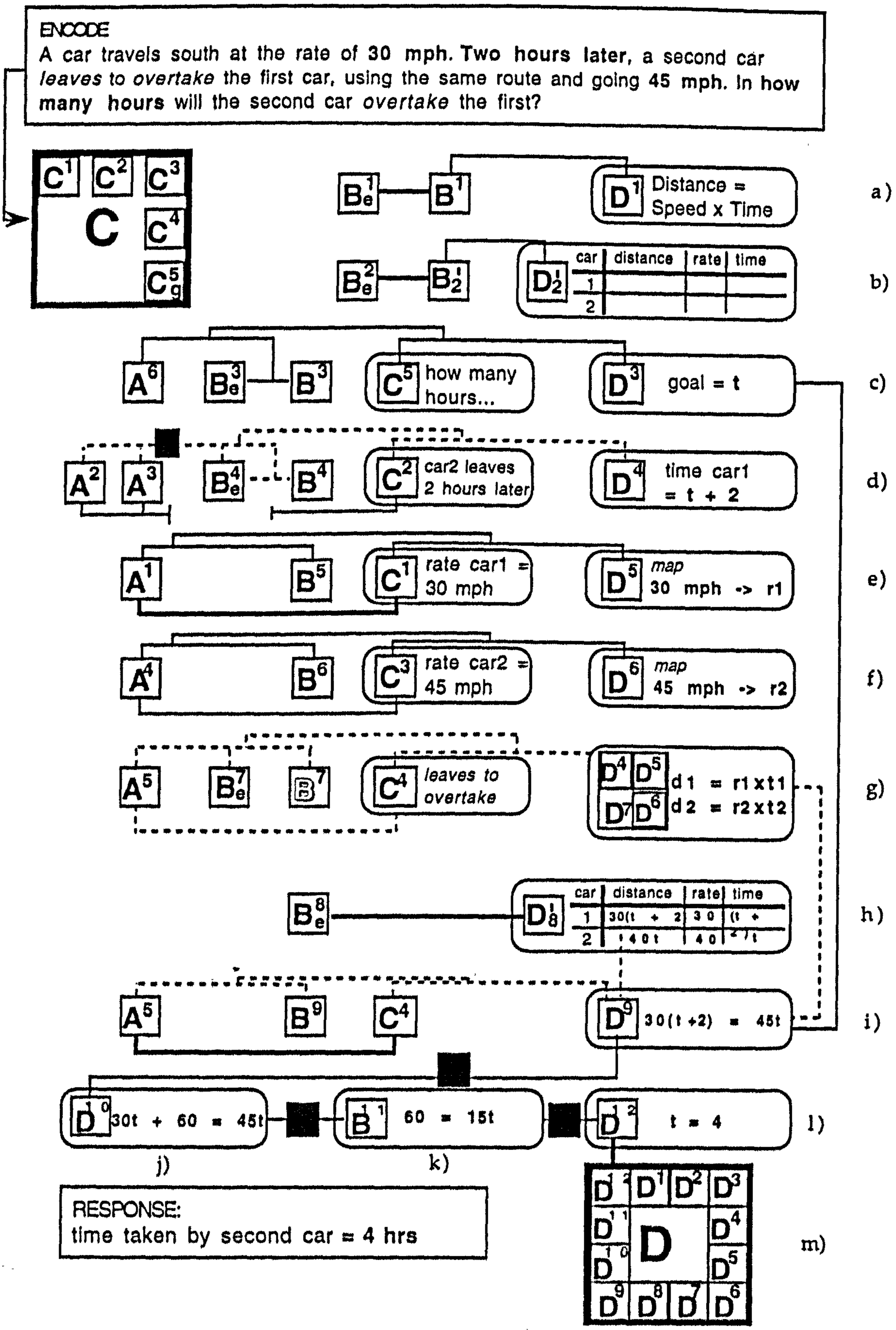


Figure 4.36. Summary figure of mapping between 'equivalent' exercise problem and source practice problem.

5.3. Mapping the unelaborated training problem onto the similar test problem

Here is the similar test problem:

> *A pick-up truck leaves 3 hr after a large delivery truck but overtakes it by travelling 15 mph faster. If it takes the pick-up truck 7 hr to reach the delivery truck, find the rate of each vehicle.*

The givens in this problem are also related by the *Distance = Rate x Time* equation. Once again the distances travelled by the two trucks are the same so the first part of the practice solution is relevant if trucks are substituted for cars:

$$Distance = Rate \times Time$$

> *Because both cars travel the same distance, the distance of the first car (D_1) equals the distance of the second car (D_2). Therefore:*

$$D_1 = D_2 \quad or \quad R_1 \times T_1 = R_2 \times T_2$$

However, the givens in the similar test problem are different from those in the source which continues:

> *where $R_1 = 30$ mph, $R_2 = 40$ mph, and $T_1 = T_2 + {}^3/_2\ h$*

Errors may arise when substituting the values in the target problem for the variables R_1, R_2, T_1 and T_2. For example, the only rate mentioned is 15 mph. Substituting that for R in the equation would lead to error. The same is true of the times; should7 *hr* or 3 *hr* replace T_2 and if so, what should be added to it? There is a further difficulty in that the first car, the one which leaves first in the practice problem, is mentioned first, but in the similar problem the truck which leaves second is mentioned first. Errors may therefore arise if the relevant car is not mapped onto the correct truck (see Ross, 1987).

Figure 4.37 presents the 'similar' problem givens with the subgoals in the solution enumerated.

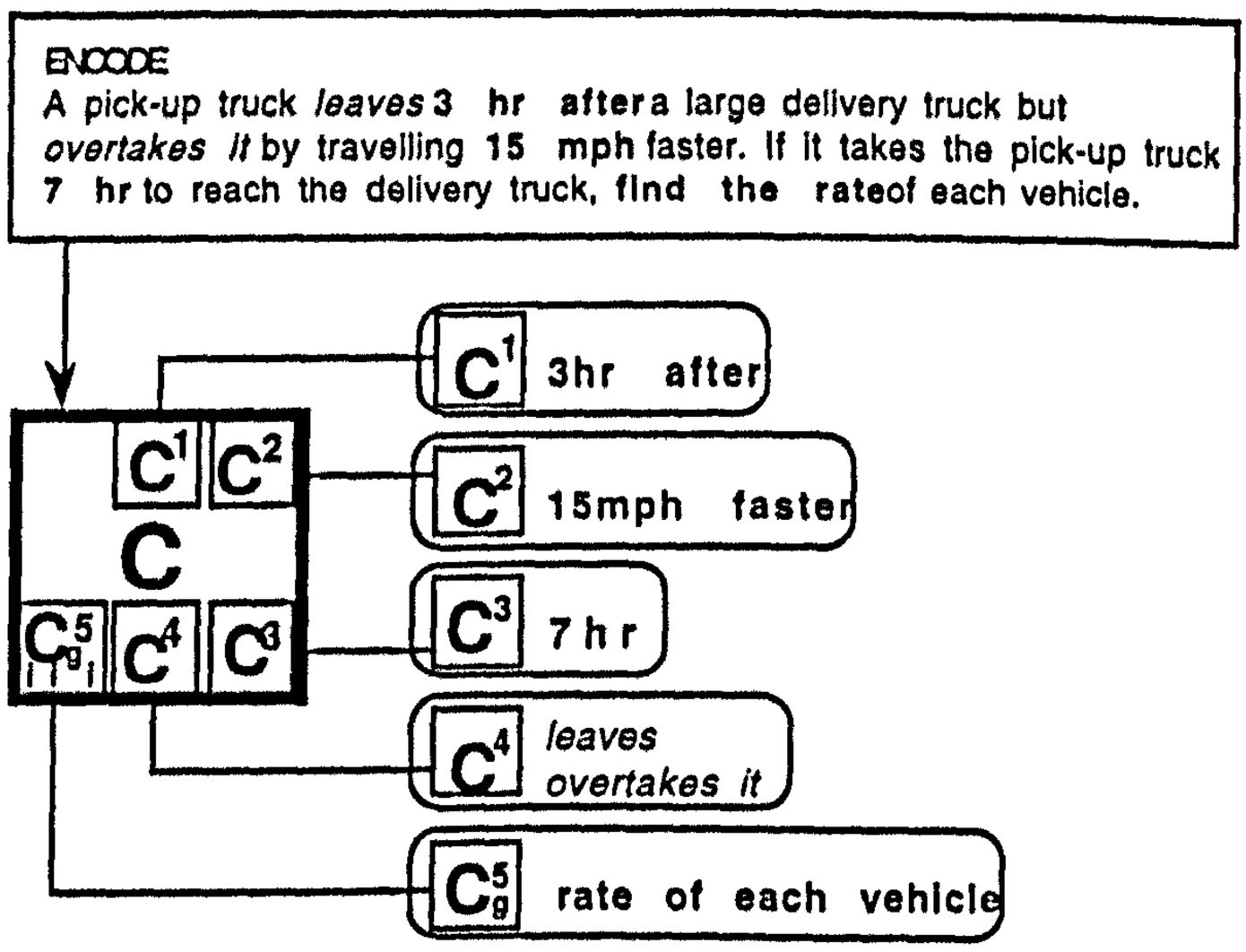


Figure 4.37. 'Similar' problem givens.

In figure 4.38, we can see that most of the A and B terms of the example problem on the left of the figure are not linked to the C and D terms on the right. Indeed the only information that the solver has is that both problems are distance-rate-time problems that involve the same equation. Nothing else in the training problem relates to the test problem.

Rows:

a, b, c) These rows represent the matches between the test and training problem. The test problem also 'inherits' the inferences that have to be made to understand the explanation of the training problem in the first place, represented by the dotted lines and black boxes in rows *b* and *c*.

d) The time taken by the first vehicle can be expressed in terms of the time taken by the second one. This goal is common to both problems. However, the match is not direct, hence the broken line from C^1. Again a lot of inferences are required to understand the earlier problem.

e) The time taken by the second vehicle is given in the problem statement (C^3 - 7 hours). Even so, solvers may still not know what to do with it if they are imitating the earlier problem since no times are mentioned in the earlier problem.

f) The problem gives a difference in rates (C^2). No differences are given in the earlier problem. Subjects are not told they have to express the rate of one vehicle in terms of the rate of the other, hence the black box between C^2 and D^6.

g, i) The two goals of the problem C^5_{gi} and C^5_{gii}, which represent the goals of finding the rate of the first and second truck respectively are not the same as the goal in the source example. To find the rate of the second car the solver has to go back to subgoal D^6 in row *f* where the rate of the second truck is given in terms of the first. The algebraic transformations in rows *h* and *j* are also different from those in the example.

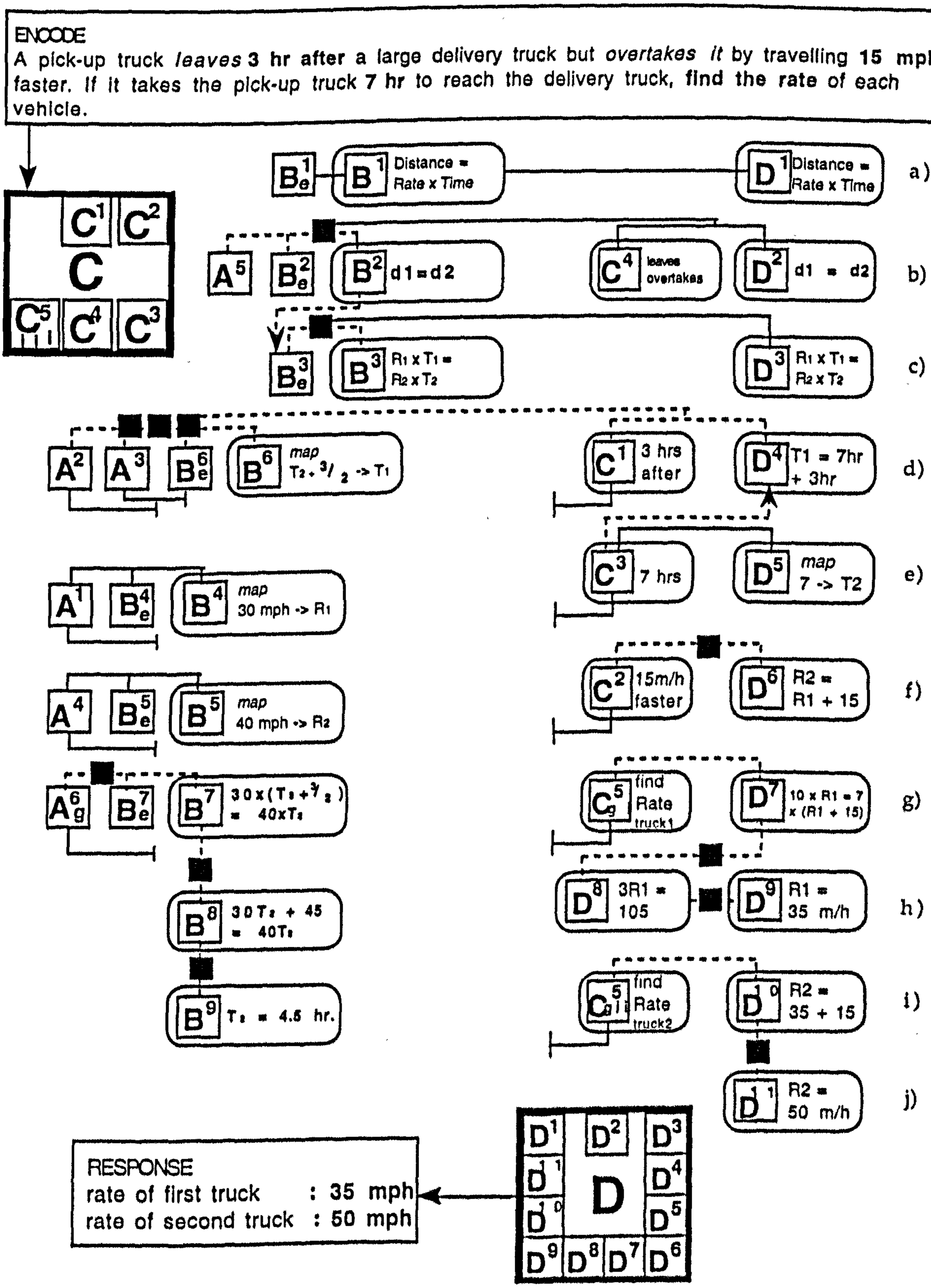


Figure 4.38. Summary figure of the mappings between the unelaborated practice problem and a similar test problem.

Even a brief glance at the summary figure 4.38 shows that none of the values of the earlier problem could be mapped onto the exercise problem (blocked lines), that a large number of inferences had to be generated (dotted lines), and no information about the relevant operators was given in this particular example (black boxes), or about what subgoals were necessary.

Given that there was little difference in the success rates for problems which were either similar or unrelated to the practice problem (despite the fact that the similar and practice problems were classified by Reed et al. as being of the same type), the analyses presented above show that *from the nature of the problems themselves* the students would have had difficulty in mapping the information given in the example problem onto the information in the target. That is, solvers are unable to imitate the source problem in order to solve the target. They were able to do this in the case of the equivalent problem where more mappings were possible and fewer inferences had to be made. As far as the subjects were concerned the practice distance problem and the similar test problem were scarcely more 'related' than the unrelated practice problem. Figure 4.38 indicates precisely that.

Indeed, classifying the problems as Distance = Rate x Time problems may lead to some confusion, because a) the problem givens should be mapped onto the equation $Rate_1 \times Time_1 = Rate_2 \times Time_2$ and not the 'distance' equation; b) solvers may not fully understand the relation between the 'distance' equation and the $Rate_1 \times Time_1 = Rate_2 \times Time_2$ equation; and c) the 'distance' equation is prominent in the explanation.

5.4. Mapping the elaborated training problem onto the similar test problem.

If the solver uses the earlier example to solve the test problem, then the mappings shown in figure 4.39 have to be made. In line *c* the goals of the two problems are different. The solver has to infer that rate should be represented by r or something similar. Another inference that has to be made is what car the r refers to.

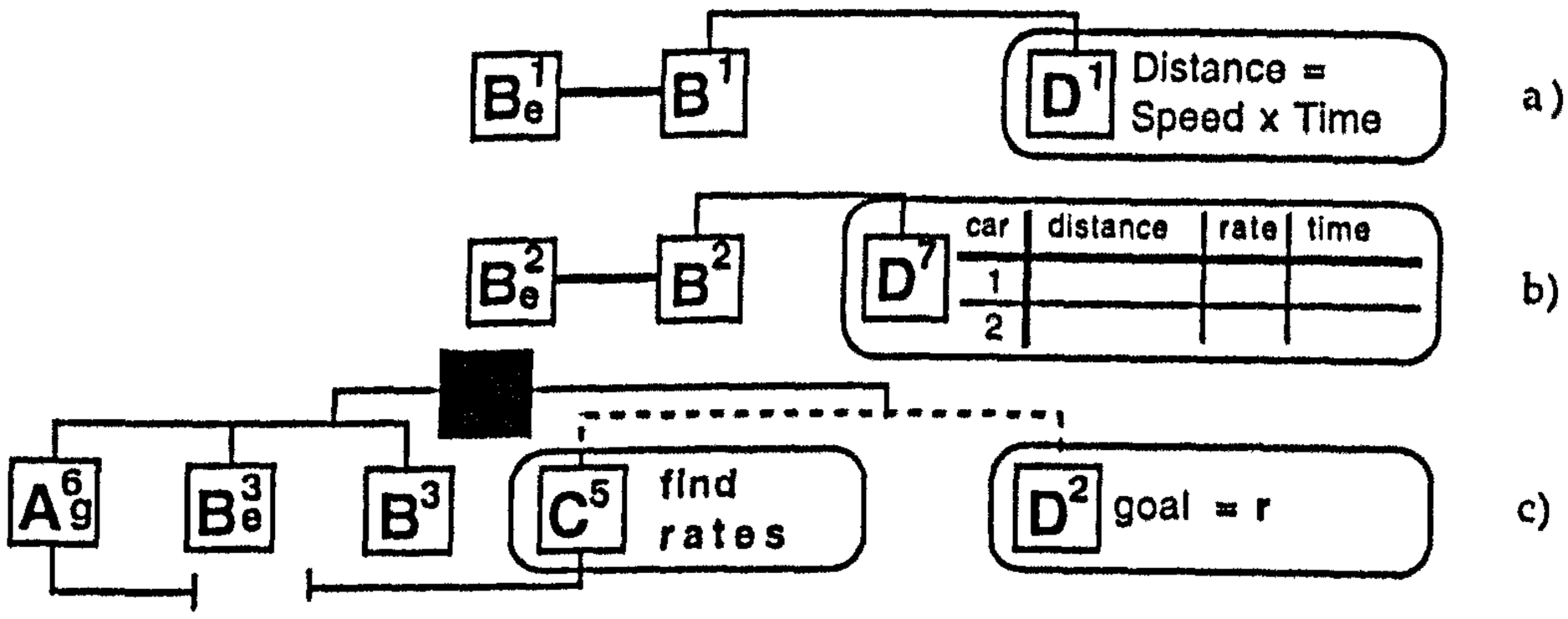


Figure 4.39. Mapping aspects of the explanation of the training problem onto the similar test problem.

In the earlier problem the subjects were not told to add the difference between the times to the time of car2 (line *d* in figure 4.40). The time of truck2 was given in the test problem but not in the example (line *e*). Nothing in the earlier problem explained what to do when the rates are unknown (line *f*).

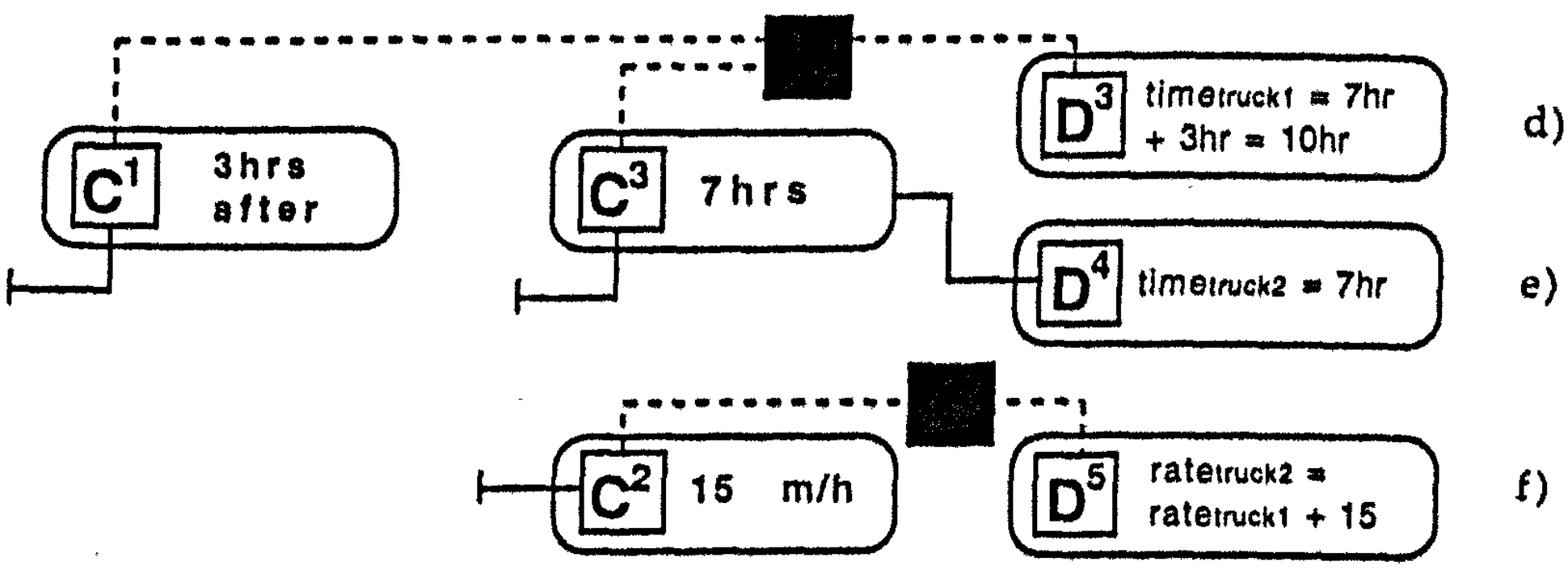


Figure 4.40. Blocked mappings between the source and test problems.

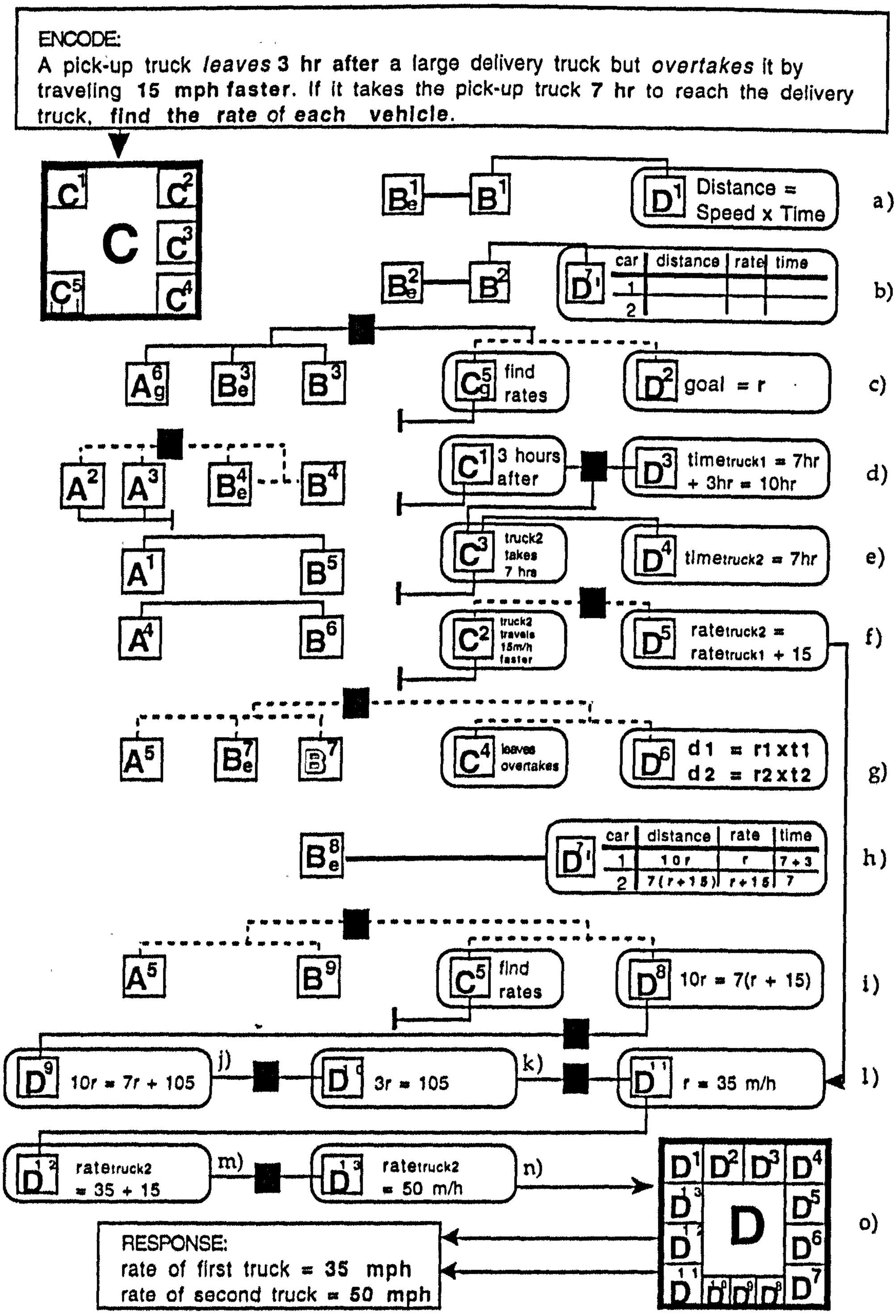


Figure 4.41. Summary figure of mappings of similar problem onto elaborated practice problem.

Figure 4.41 shows that there are difficulties in mapping and in making inferences about what elements in the example problem are relevant and what operators should be applied. Given the number of blocked access lines from the C terms to the earlier problem, the number of black boxes and dotted inference lines, it is apparent that there is very little of the earlier problem that is of any use to the solver. The practice problem and the test problem are therefore different problems.

Paradoxically, where students had available to them an elaborated solution to work from when solving a 'similar' problem, they were obliged to make at least as many inferences as in the experiments where only an unelaborated solution was provided. A larger number of subgoals had to be generated, since a greater number of steps was required. Despite the fact that the equation is the same as the one in the source problem, and that the solver is required to construct a table to aid in instantiating the variables, only 22% of students successfully solved the problem. When students were presented with an unelaborated solution 17% successfully solved it. Although no data are available in the original paper, it seems unlikely that the difference between these results is statistically significant. Certainly the improvement is marginal. Where the elaborated solution to the previous problem was shown and then removed, the worst results of any in the experiments were obtained with only 6% correct; when the subjects were presented with an unrelated practice problem, the test problem was successfully solved by 14% of students. According to Reed et al. the low level of performance was due to students' inability to generate the correct equation rather than an inability to solve it when presented with similar problems and an elaborated solution. But then, in the condition where students were obliged to rely on their memory of the solution, they had to make more inferences and generate more subgoals from memory than in the unelaborated condition.

It would appear from this analysis that the number of inferences that had to be made from the source problem caused difficulties for the students. The representations of the problems show exactly where no information or guidance was provided as to how mappings could be made from one problem type (finding the *time* taken by *one* vehicle) to another (finding the *rates* of *two* vehicles) and this despite the fact that the same equation is used to solve both types of problem.

6. Discussion

According to Reed et al., the purpose of the three experiments examined here was 'to investigate how well students could use the solution of an algebra word problem to solve other problems belonging to the same category.' The category (*distance-rate-time* problems) is one which has been imposed by the authors and not one which is necessarily understood by the students (see also Anderson, 1990; Hall, Kibler, Wenger, & Truxaw, 1989). Close variants of the example problem, where the time taken by one vehicle had to be found, tended to be solved by most students when an elaborated solution was provided for study. Distant variants proved too difficult for most. Therefore, as far as the students are concerned, the similar and the practice problems were *not* in the same category. Indeed, from the analysis of the mappings between the example and distant variants given above one might reasonably predict that there was likely to be little difference in success rates between the unrelated problem and this particular so-called 'related' problem.

When Reed et al. state that the earlier problem had to be 'slightly modified' they made a judgement that only a relative expert can make. No mappings can be made between the features of the source and those of the 'similar' target. To solve the similar problem subjects are obliged to make a large number of inferences and have a great deal of difficulty in doing so. The analysis of the tasks with which they were presented has identified where these inferences and blocked mappings lie. Most subjects did not have a representation of the earlier solution which was adequate enough to allow them to manipulate it in order to solve the target. Instead they attempted to imitate the previous solution and failed because they did not have enough information to do so.

In their experiment 4, Reed et al. also noted that a large number of matching errors occurred when students used a complex practice problem to solve a simple test problem. Solvers will have difficulty using such a source, since any variation will be a source of error when subjects are attempting to imitate it. The inadequacy of the explanations is alluded to by Reed at al. when they make the point that 'subjects had considerable difficulty in specifying the relations among variables'. The analysis presented here has demonstrated that the problems themselves did not provide enough information to relate the problem givens to the subgoals in the solution.

Where the solver has an isomorphic earlier problem in view along with a complete set of procedures for solving problems of this particular type, the solver should be able to make adequate mappings between the example problem givens and its associated solution as

well as from the example problem givens and the test problem givens. However, as has been demonstrated, the elaborated explanations in Reed at al.'s study are incomplete and still leave the solver with the necessity of generating inferences. When solvers are required to use those incomplete explanations to try to solve a 'similar' problem then the solution rate drops dramatically. This suggests that the solvers' representations were very inflexible and that the mappings were taking place only at a surface level. That is, the help provided by the elaborated solution allowed the solvers to imitate the earlier example by mapping its surface features onto the target. Whenever the earlier solution method had to be 'slightly' modified, most of the students were incapable of doing so because they could not find the necessary correspondences between the problems. In other words, the practice problem was different from the one they were required to solve.

Although much has been written about the difficulties people have in analogical problem solving, not enough has been written about how to improve the texts in the first place to increase transfer between problems. Now that we have identified the areas where inferences have to be made in these algebra problems, the next stage is to find out what happens when the inferences are removed. This is the subject of the next section.

Chapter 5 UNDERSTANDING, IMITATION, AND PROBLEM PRESENTATION

1. Introduction

In this chapter I describe a number of experiments looking at problem understanding, imitation and the effects of different problem presentations. The first of these - problem understanding - was examined by comparing subjective and objective measures of 'understanding' to see how well they correlated with each other. The studies were designed to find out if subjects were able to assess their own understanding of problem explanations and, if so, how this can help us in the design of expository text; experiments 1 and 3 examine this issue.

Experiments 2 and 3 provide an empirical test of the interpretation theory as it is applied to the text of word problems. The analyses in Chapter 4 showed where inferences remained in the distance problems used in Reed, Dempster and Ettinger (1985) and therefore where the solver would have difficulty when solving both a close variant of the problem and a distant variant. The experiments in this section are designed to justify those claims.

2. Problem understanding: Experiment 1

To investigate the relation betweeen understanding and problem solving, an experiment was carried out using problems in PROLOG. In Chapter 2, understanding was defined in

terms of 'flexibility' or the ability to manipulate and adapt learned material. Hiebert & Lefevre (1986) maintained that 'powerful problem solving is impossible without understanding ... [which involves] ... the processing of the original problem presentation to construct a meaningful internal representation that can be manipulated by the solver in order to produce the desired result.'

The study presented here sought to find out whether subjects' assessment of their own understanding would correlate with their problem solving performance. Problem solving performance on a transfer task is often taken to be a measure of analogical problem solving ability, which in turn depends on the solver having an adequate mental representation of the example problem used as an analogue. If solvers do have such a mental representation then they can be said to 'understand' the problem in the sense defined above. However, in Chapter 2, I argued that novices and beginners often do not have a representation of a source problem that is 'complete' enough to allow them to reason analogically. Instead they tend to use imitative problem solving.

Previous research has shown that subjects do not always have a clear idea of what they understand (Chi, Bassok, Lewis, Reimann, & Glaser, 1989; Ferguson-Hessler & Jong, 1990). Ferguson-Hessler and Jong, for example, found that 'poor' students said 'everything is clear' three times more often than good students whereas their performance showed that this was not the case. In analysing the study processes of students studying physics texts, they also found that poor performers processed declarative knowledge more than good performers, who concentrated more on situational and procedural information.

Hiebert & Lefevre (1986) distinguished between 'primary' and 'reflective' understanding. Primary understanding occurs when the subject understands the relations between elements in a new domain at the same level of abstractness as, or at a less abstract level than, the information being presented. This type of understanding is highly context specific. Reflective understanding is at a more abstract level. It occurs when subjects recognize the deeper structural features of problems.

The distinction can be understood by looking at this PROLOG rule:

```
has(_X  _Some-feature) if
    isa(_X  _Category-member) &
    has(_Category-member  _Some-feature).
```

A primary understanding of the rule would involve an understanding of its declarative structure. That is: 'X has some feature if it belongs to a category and that category has the

feature in question.' A 'reflective' understanding might include a procedural representation of the rule which may allow the subject to generate inferences: 'To find out whether X has some feature, first find out if it belongs to a category and then find out if that category has that feature.' A deeper understanding of the rule might also include the realization that it calls itself within itself with new variables (Category-member instead of X), and, more importantly, that there is no halting condition; for example, it should have something like 'has(_X _Some-Feature).' before the above rule.

The experiment presented here set out to compare subjects' estimates of their own understanding with their performance on a task that required a fairly deep understanding of the examples presented. If students concentrated more on the surface features of the problems we would expect their understanding of the material to be at a 'primary' level. If they understood the new material in the sense of trying to integrate it with their prior knowledge, or if they made some attempt to evaluate the new material, then their understanding would be at a more 'reflective' level and ought to correlate with problem solving performance on a variant of the example problems presented.

2.1. Method

2.1.1. Subjects

The subjects were 16 adults following a Cognitive Psychology course in the Open University. The subjects were approaching the end of the one-year course and had completed a course unit on programming in PROLOG.

2.1.2. Design and Procedure

Subjects were shown two examples of a PROLOG database divided into FACTS and RULES. The RULES were labelled A, B and C. The first involved a hierarchy of animals (the full text of the questionnaire is reproduced in Appendix 1). The first part of the FACTS database gave examples of categories of animals such as:

isa(mammal animal)

isa(dog mammal)

isa(Butch dog)

The second part gave features of some animals:

has(mammal fur)
has(fish gills)
has(horse mane)

The PROLOG rule was designed to show inheritance, such that, if something was an instance of some category, then it inherits the same features as the superordinate category:

A

```
has(_Something _Some-feature) if
    isa(_Something _Category-member) &
    has(_Category-member _Some-feature).
```

(The syntax used for PROLOG was an early MacProlog variant that differed from 'Edinburgh' syntax.)

The second example showed a train timetable which included the destinations and departure points of trains in the FACTS section using the predicates 'departs' and 'arrives', and two RULES. The first states that *two stations are directly linked* if the same train leaves station1 and arrives at station2:

B

```
direct-link(_Station1 _Station2) if
    departs (_Train _Station1) &
    arrives(_Train _Station2).
```

The second and third rules state that two stations are *connected* if they are either directly linked or if there are directly linked stations on the way:

C

```
connected(_Station1 _Station2) if
    direct-link(_Station1 _Station2).
connected(_Station1 _Station3) if
    direct-link(_Station1 _Station2) &
    connected(_Station2 _Station3).
```

For the first two sheets, subjects were given the following instructions:

Read this sheet and, in the boxes under column 1, write a number from 1 to 5 to show how well you think you have understood the statements or the PROLOG clauses according to the following code:

5: Understand perfectly 4: Understand reasonably well 3: I think I have got the gist

2: Don't quite understand 1: Don't understand at all

There were boxes alongside the database and rules, and alongside the natural language explanations of them. The third sheet contained a database of types of food and the food preferences of vegetarians and carnivores. Subjects were asked to imagine that they had to write a rule that would represent the food preferences of omnivores:

Food Preferences

FACTS

Here is a database of food 'facts':

isa(meat food).
isa(poultry food).
isa(vegetables food).
isa(pulses food).
isa(grain food).
isa(fish food).
isa(carrot root).
isa(potato root).
isa(lettuce leaf).
isa(cabbage leaf).
isa(beef meat).
isa(chicken poultry).
isa(root vegetable).
isa(leaf vegetable).
isa(trout fish).
isa(salmon fish).
isa(lentils pulses).
isa(apple fruit).
isa(rice grain).

Here is what vegetarians and carnivores eat:

eat(vegetarians vegetables).
eat(vegetarians pulses).
eat(vegetarians fruit).
eat(vegetarians grain).
eat(carnivores meat).
eat(carnivores fish).
eat(carnivores poultry).

1) Now suppose that you were given the problem to write a rule or rules to represent what it is that omnivores eat. Think about this for a few minutes and then look at the other two sheets with the **train timetables** and the **hierarchy of animals** and, in the second column of boxes, tick anything that you think would help you solve the problem.

2) Choose, between rules A, B and C on the other two sheets, which one you think would be the most useful to you in solving the problem and write your answer in this box: ☐

3) Which of these do you think would be the most likely first line of your rule (tick the appropriate box):

☐ eat(_X food) if ...;
☐ eat(omnivores _X) if ...;
☐ eat(_Omnivores _X) if ...;
☐ eat(omnivores food) if ...

4) Are you acquainted with Prolog or any other artificial intelligence language outside the D309 course? **Yes / No**

5) (Optional) Try writing the rule. If you do so, could you please tick everything that you refer to on these sheets while solving the problem, adding a tick every time you do so.

This was the subjects' first exposure to recursive functions. No attempt was made to relate the natural language explanations of the database and the rules (represented by A1 and A2 in figure 5.1 for the animal hierarchy and the train timetable respectively) to the PROLOG code (B1 and B2). Nor was there any attempt to relate the two examples to each other. Since no explicit explanation is given of the relations between the statements and the rules, the A and B boxes are linked by dotted lines to show that the reader has to make inferences to understand the relations.

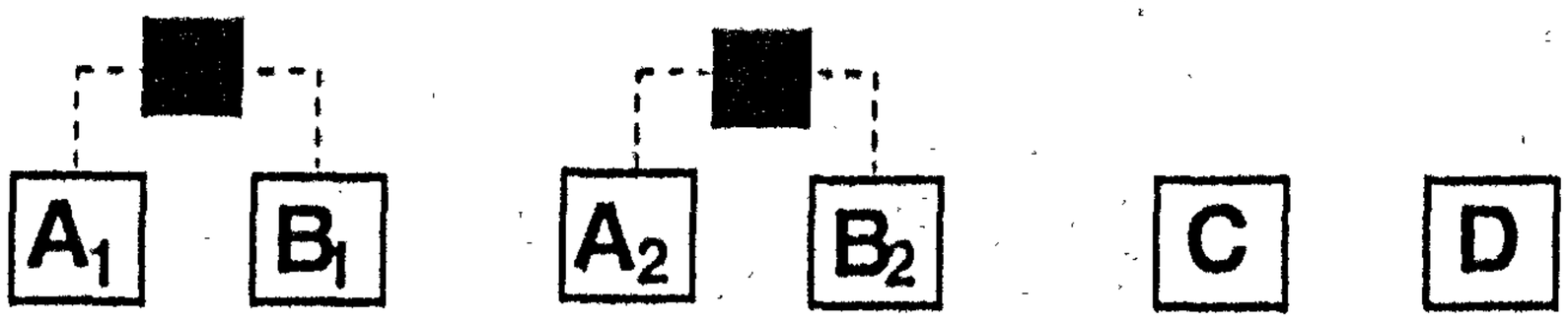


Figure 5.1. An analysis of the text of the PROLOG questionnaire examples.

One solution to the test problem is structurally similar to rule C (the *connected* rule) they were shown in the examples:

```
eat(omnivores _X) if
        isa(_X food).
eat(omnivores _X) if
        isa(_X _Y) &
        eat(omnivores _Y).
```

A declarative reading would be 'Omnivores either eat food or they eat some item and that item is an example of something that they would eat.' The rule differs from the *connected* rule in that the variables are manipulated differently. Unlike the *connected* rule, however, it does not rely on an earlier rule the way rule C relies on rule B, the *direct-link* rule. In that sense it is simpler than the rules in the railway timetable example. The second part of the *omnivores* solution is also similar to rule A in the animal hierarchy example, but the latter is structurally simpler since there is only one part to it.

2.2. Results

Figure 5.2 shows the average rating scores under different headings for each part of the example explanation, as well as ratings for understanding the PROLOG facts and rules themselves. The headings were:

E.D1	Explanation of Database 1
E.D2	Explanation of Database 2
E.R1	Explanation of Rule 1
R2a	The first part of Rule 2
D1	PROLOG Database 1
D2	PROLOG Database 2
R1	PROLOG Rule 1
R2b	The second part of Rule 2

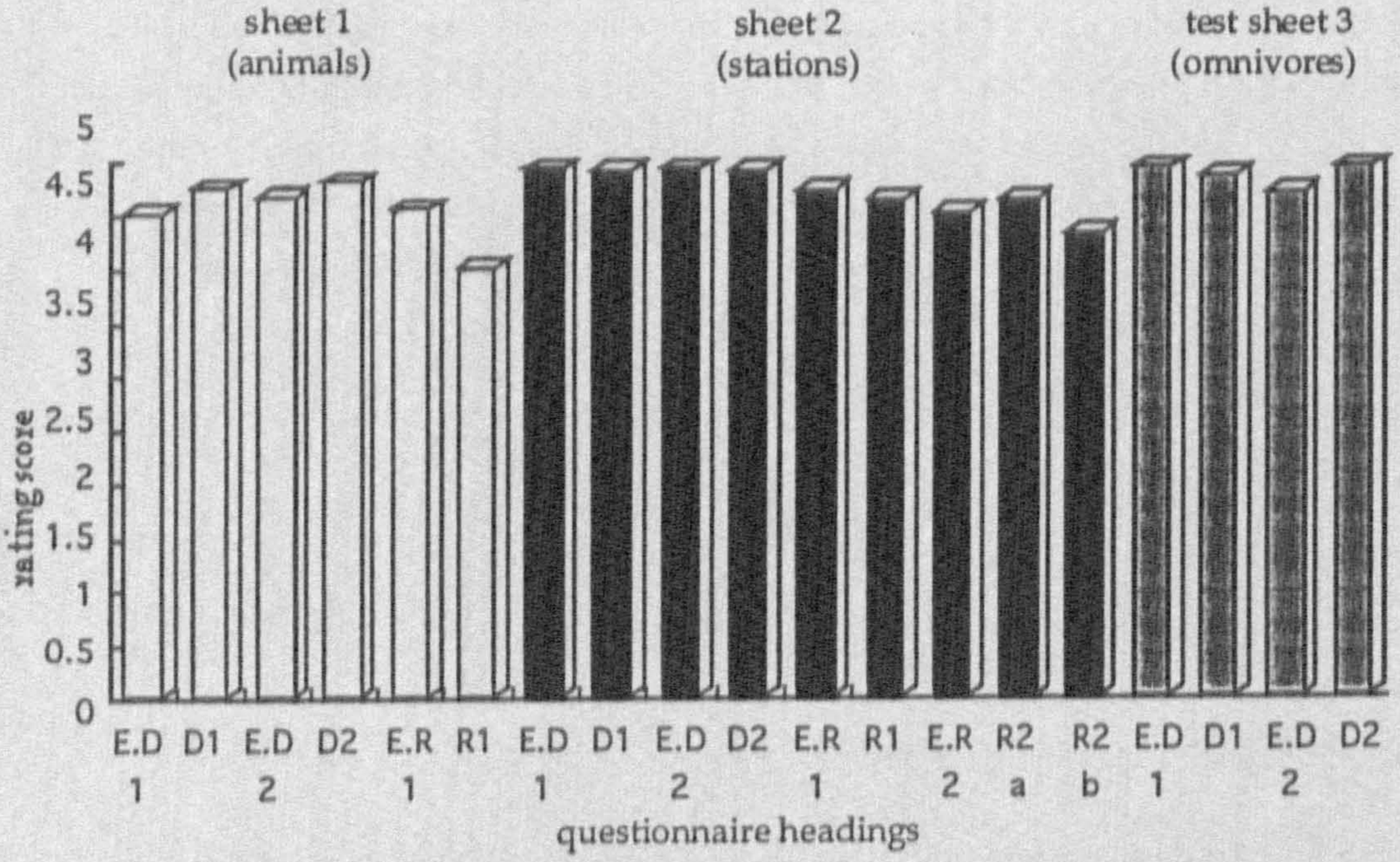


Figure 5.2. Average rating scores for each item in PROLOG questionnaire.

In general the subjects in this experiment claimed to have a high degree of understanding of the example problems presented with an average rating of 4.7. This was in spite of their being no explanation of the relation between the statements of the problem and the PROLOG code used; nor was there an explanation of how the examples could be used to solve other problems of the same type. Nevertheless, their assessment of their own understanding ranged from 4.00 (for the first PROLOG rule presented) 'Understand reasonably well', to 4.94 , 'Understand perfectly.'

2.2.1. Programs generated by subjects

Although it was an optional activity, most subjects attempted to write the code for the omnivore problem. No-one managed to generate the correct code for the problem[1]. The subjects' responses are listed below (along with their comments where they made any):

S1
```
eat(omnivores _X) if
        eat(carnivore _X).
eat(omnivores _X) if
        eat(vegetarians _X).
```

S2
```
eat(omnivores   ) if
        isa
```

S3
```
eat(omnivores _X) if
        isa(_X food).
```

S4
```
eat(omnivores _X) if
        isa(_X food).
```

S5 *none*

S6
```
eat(omnivores _X) if
        isa(_X food).
```

S7
```
eat(omnivores _X) if
        isa(_X food).
```

S8
```
eat(omnivores _X) if
        isa(_X food).
```

---

[1] *Due to a possible misinterpretation of the question, a further study was carried out in which 10 subjects had to write the same rule as above. This time, however, they were told to ensure that the query* eat(omnivores carrot). *would succeed. The results were identical to the study described here - no one solved the problem.*

```
eat(omnivores _X) if
        isa(_X _Y) &
        isa(_Y food).

eat(omnivores _X) if
        isa(_X fruit).

eat(omnivores _X) if
        isa(_X _Y) &
        isa(_Y _Z) &
        isa(_Z food).
```

S9	eat(omnivores _X) if isa(_X _food) & has(_food roots).	
S10	eat(omnivores _X) if isa(_X food).	
S11	eat(_X FOOD) IF ISA CARNIVORE & ISA VEGETARIAN	
S12	eat(omnivores _X) if isa(_X food).	'This seems too simple, but I can't think of anything else.'
S13	*none*	'Sorry - in theory I understand Prolog, in practice I do not think I do!'
S14	eat(_X food) if isa(_X omnivore) & eat(_X omnivores food).	'This [code] has been tried without consulting any other information except my memory.'
S15	*none*	
S16	eat(omnivores _X) if	

75% of the subjects chose the correct first line of the rule: *eat(omnivores _X) if....* Most wrote a program whose declarative meaning corresponds to their real-world knowledge; that is, omnivores will eat anything that is food: *eat(omnivores _X) if isa(_X food).* However, the query: *eat(omnivores carrot).* among others, would fail since there is no pattern matching *isa(carrot food).* S1's response states that an omnivore would eat anything that a carnivore or vegetarian would eat. However, the query *eat(omnivores carrot)* would fail for the same reason. The subjects did not realize that the database of things that carnivores and vegetarians eat contained items which were 'mid-way' between the superordinate category 'food' and the specific examples such as 'carrot,' 'fruit,' 'fish,' etc. The database of food preferences is therefore incomplete, since there is

nothing to state that *carrot* is an example of a category of things that vegetarians or carnivores eat. Subjects' solutions were therefore not based on the structure of the PROLOG database they were given.

Only one subject, S8, tried to cover all possibilities. It seems likely that this subject realized the limitations of *isa(_X food)* used by itself and tried to go through all possible cases - a sort of 'manual recursion' - even though both previous examples in the questionnaire contained a recursive procedure (which was not explained) that would have worked through the hierarchy. S8 and S12 could not see how any of the previous examples could help.

S14 attempted a recursive solution based on rule A. Unfortunately the last line of the code contained three arguments and could therefore not match anything in the database.

2.3. Discussion

The near perfect 'understanding' claimed by the subjects did not help any of them correctly solve the exercise problem. There is therefore no correlation between the subjects' assessment of their understanding and their performance on a transfer task. One solution to the *omnivores* problem involved adapting the slightly more complex timetable solution or the simpler animal hierarchy one. Despite subjects' statements that they understood both examples almost perfectly, they did not understand them sufficiently well that they were able to adapt them to solve the target problem. That is, they did not see that the database was incomplete; it contained the information that a carrot was a root, that a root was a vegetable, and that a vegetable was food. It did not state explicitly that a carrot was food. Subjects had to write a rule that would work through the hierarchy of *isa* rules to determine that. They also seemed unaware that the point of the rules in the earlier problems was to allow just that. That is, the rules in the previous examples were designed to allow an object to 'inherit' any characteristics from higher up the hierarchy of relations as in the animal hierachy example, or to work through all intermediate stages, as in the *connected* rules, when there was no 'direct' route to an answer. This was the kind of understanding required to solve the test problem.

Since there was no correlation between the 'subjective' and the 'objective' measures of the subjects' understanding, we can characterize the type of understanding that subjects have as 'primary'. That is, their understanding was at a superficial level. They understood the relations between the superficial elements of the examples but not at a level 'deep' enough to be able to abstract out the general principles underlying them and apply them to a new instance of the same type of problem.

It would, however, be useful if we could produce a context in which the explanation of an example did lead to a correlation with what students said they understood and how well they performed. The current experiment does highlight that we cannot rely on people's own estimates of how well they comprehend textbook explanations. Fergusson-Hessler and Jong found that 'poor' performers tended to concentrate more on declarative knowledge than procedural knowledge. 'Good' performers applied more 'deep' processing of information. In the present experiment, subjects had little to go on other than the meaning of the database and rules. There was no explanation of how the rules worked, for example. However, it would be unreasonable to label all the subjects in this experiment as 'poor' performers; it is much more likely that the lack of correspondence between subjects' assessment of their understanding and their ability to manipulate an example problem was due to the incompleteness of the text rather than any kind of failure on the part of all the subjects. It may be that improving the quality of the textual explanations would lead to a more accurate assessment of understanding on the part of the subjects. This topic is addressed in experiment 3.

3. How can problems be made easier to solve? Experiment 2

In Chapter 4, the analysis of some of the algebra problems that Reed et al. (1985) gave their subjects indicated areas where the student has to make a number of inferences in order to solve various transfer problems. Reed et al. claimed to have removed most of the inferences in their original problem explanation by representing the practice problem in the form of a table and giving more information about how the elements of the problem statement (30mph, 45mph, difference in departure times, etc.) related to the solution equation. If the student used a table which could be filled with values derived from the problem statement, then it could be used to fill in the variables of an equation for this type of problem. Reed and Ettinger (1987) examined the use of tables in more detail. They found that subjects had difficulty filling in the table but could readily generate the necessary equation when given a completed table. They also found that subjects could, to some extent, use tables to adapt problems in order to solve more distant variants, but only when completed tables were provided; there was no transfer to isomorphic problems when the completed tables were not provided.

Chapter 3 showed how a simple word problem could be analysed into A and B terms and the relations between them. Where these relations are not completely stated the problem's usefulness as an exemplar of a problem category is limited. Its usefulness is similarly compromised if no information is given about how an example relates to further problems of the same type (Conway & Kahney, 1987). Some aspects of the relation between example and test problems were examined by Novick and Holyoak (1991). Using algebra word problems they looked at the effects of giving subjects specific numerical mappings for transfer problems. For example, in the 'number mapping hint' condition subjects were presented with hints such as: 'the 12, 8, and 3 in the band problem are like the 10, 4, and 5 in the garden problem'. Transfer success was much more likely to occur with number mappings than if the subjects were given 'concept mappings' such as: 'your goal in this problem is to arrange the band members into rows or columns so that each row (or each column) has the same number of people in it, with no-one left over. That's like the goal you had in the garden problem of grouping plants into different types so that there were the same number of plants of each type, with no extra spaces left in the garden.' They found that the numerical mappings were a necessary (but not sufficient) prerequisite for transfer. The difficulty came when subjects had to *adapt* the procedure to solve a transfer problem.

Problem adaptation takes a variety of forms; Novick and Holyoak list three of them:

1) *Substitute numbers from the test problem into the source operators.* Elements of one problem have to correspond to elements in the other. An example would be substituting, say, 30 mph in one problem for 40 mph in another in distance-speed-time problems. Failure to map the correct numbers led to what Reed et al. called a 'quantity error', and using a number in a source problem without changing it in the target led to a 'matching error.' This form of adaptation is not a major source of difficulty when the problems to be solved are 'literally similar.'

2) *Postulate new test-problem elements not described in that problem.* This occurs when the target is a distant variant of the source problem. If the source contains 'time taken' but the target does not, or if the target gives 'rates of travel' but none are given in the source, then the subject will have to generate new test problem elements. If subjects attempt to imitate a source problem to solve a distant variant, they will not be able to map values across since the relevant values do not exist. Subjects will fail to solve the target problem unless they recognize the relation between the source and target at a more abstract level than the level of surface features. If they can do so, then they can use the underlying solution structure or principle to infer new problem elements.

3) *Generalize source procedure in ways that preserve the essential structure of the procedure.* If the procedure involves generating an equation, say, in an example, then the equation has to have the same form in any test problem. Failure to preserve the problem structure would lead to what Reed, Dempster & Ettinger call a 'frame error', where the form of the equation is wrong.

The types of manipulation of a source described in (2) and (3) above are those which create the most difficulty. They can only come about if the subject understands the source well enough to be able to manipulate it. The type of adaptation outlined in (1) is simply one of mapping corresponding surface feaures from one problem to another; it poses the least difficulty for IPS.

To get round the difficulties people have with adapting a solution procedure to solve variants of a problem type, the procedure should give as much relevant information as possible about how to solve problems of that type. This can only be achieved when the example problem is presented at a level general enough to apply to a range of such problems *and* specific enough to show how mappings between problems can be made. Novick & Holyoak gave subjects the 'number mapping' with the exercise problem as a hint about how to *use* them. Exercise problems in textbooks generally don't do that. Nor will students find such specific information in many types of problems they will normally be required to solve. For this reason the examples provided should be as inclusive as

possible, in the sense used by Reed, Ackinclose & Voss (1990). That is, they should be more complex than the early test problems that students might be expected to solve.

Novick and Holyoak also found that adapting the example was correlated with mathematical expertise and not analogical reasoning ability as tested by conventional IQ type tests. This should not really be surprising. Analogical reasoning involves the ability to apply knowledge which a person already has. This is Newell's definition of intelligence (described in Chapter 2). In a relatively new domain people will vary in the amount of prior knowledge they can bring to bear on a problem. Those with greater mathematical expertise have ipso facto more knowledge, which will help them infer the necessary mathematical operators.

To summarize: novel problems can be made easier to solve if:
a) as much relevant information as possible is provided at the outset;
b) this information is organized at different levels from the general to the specific;
c) the solution procedure includes information about how it can be adapted to solve variants of the problem type; and
d) subjects possess enough mathematical expertise to adapt the procedure where necessary.

3.1. Empirical evidence for the usefulness of the interpretation theory

The interpretation theory allows for a fine-grained analysis of a text. We can therefore be very specific about which parts of the example's solution procedure will cause difficulties for solvers. The figures which follow represent parts of the solution procedure for the practice distance problem used by Reed et al. (1985) (for clarity some of the figures from Chapter 4 are reproduced in this section). They indicate specific areas where a solver is likely to commit an error. From the information in those figures we can make some predictions about where students are likely to have problems in using this example as a basis for solving other problems of the same type.

3.1.1. Predicted sources of difficulty in understanding the example problem

In this subsection I will briefly review the specific areas where students are likely to have difficulties in understanding the example distance-speed-time problem. The next subsections deal with difficulties in mapping it to a close and distant variant.

The analysis of the Reed et al. elaborated practice problem in section 4.3.2 of Chapter 4 showed that there were several areas where a number of inferences had to be made. These are shown in figures 5.3 and 5.4 (which form part of the summary figure 5.5).

a) The practice distance problem states: 'we let t represent the number we want to find and enter it into the table. The first car then travels $t + 3/2$ hr because it left $1\,1/2$ hours earlier.' The explanation does not say where the $1\,1/2$ hours came from, hence the dotted lines and black box from the A terms to B^4e in figure 5.3. There is no indication why $1\,1/2$ hours was changed to $3/2$ in B^4 which is why there is a black box between the explanation B^4e and the subgoal B^4.

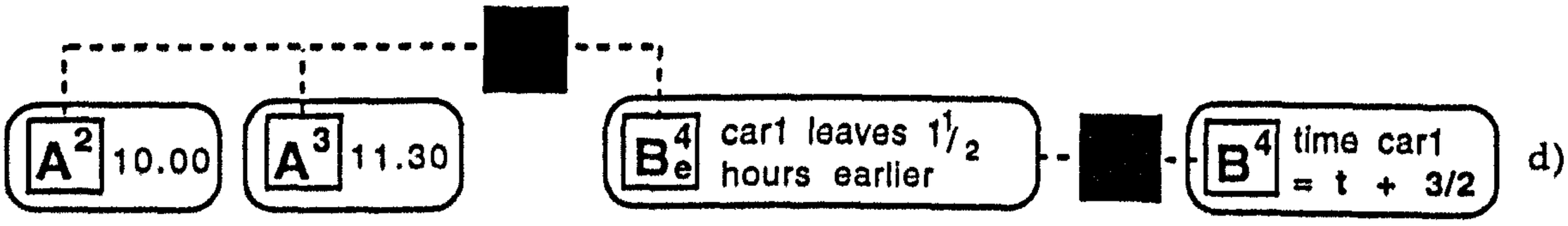


Figure 5.3. Predicted difficulties in understanding the differences in times in the practice problem.

b) The text of the example problem tells the subjects to 'represent distance by multiplying rate and time for each car' (box B^7e in row *g* of figure 5.4). It does not say what this means in terms of an equation. For this reason subjects may have difficulty mapping from the problem givens to the relevant equation. In figure 5.4*g* the B is in shadow format because the equations referred to in the text are not explicitly stated.

c) The table, taken from Reed et al.'s explanation, contains six boxes (row *h* in figure 5.4), representing the distance, rate and time for each car. The relevant equation has only 4 variables *rate x time* for each car (figure 5.4*g*) which are not explicitly stated in the explanation. Students may find difficulty in relating the distance boxes to the rate and time boxes in the table (figure 5.4*h*) since all they have to go on is the statement in box B^7e. Furthermore, and most importantly, the relevant equation is *rate1 x time1 = rate2 x time2*. The only equation given in the example is *distance = rate x time*. Generating the equation in B^9 (row *i*) will therefore be difficult for some students.

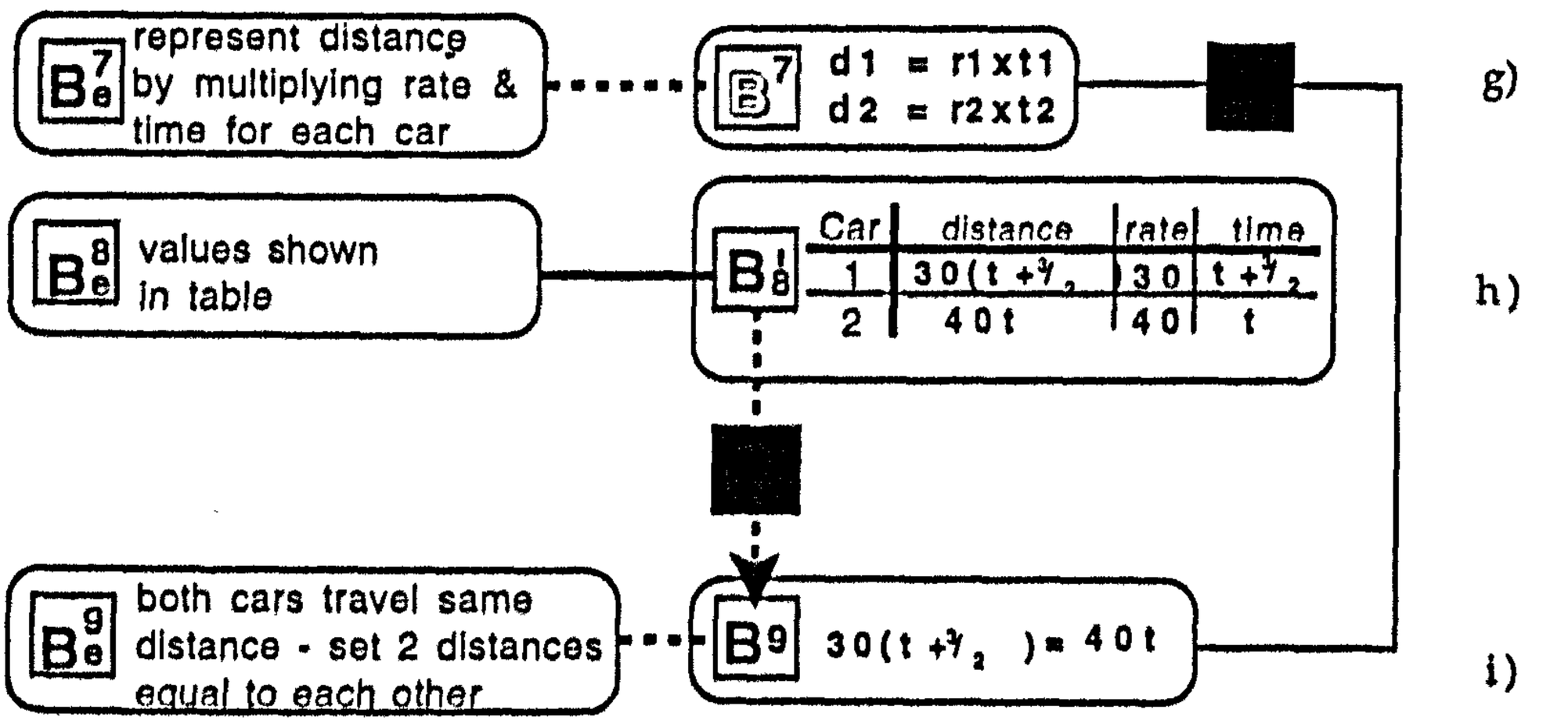


Figure 5.4 Predicted difficulties in mapping elements in the problem to the table and the relevant equation.

We can therefore predict that students will have difficulty in

i) understanding the origin of the $^3/_2$ hours

ii) accessing the relevant equation;

iii) mapping the six elements in the table to the four in the equation;

iv) filling in the distance boxes in the table.

When students attempt an exercise problem which is a close variant of the practice problem, we can predict that, where students do show any misunderstanding, it will be in those places marked by the inference lines and black boxes in figure 5.5*d* to *i*. Thereafter, they are obliged to rely on their prior knowledge of mathematics to understand the transformations that take place to arrive at the overall solution (rows *i* to *l*). The focus of this study, however, is on the students' ability to generate the equation in row *i*. If they can successfully substitute elements in the target for elements in the source to generate the equation, then solving a close variant should not cause much difficulty. So our first hypothesis is:

Hypothesis 1 The major difficulty in solving a close variant of the problem is one of mapping corresponding values. Where the transfer problem is literally similar, we would not expect mapping to pose a particular problem when solvers are imitating the source.

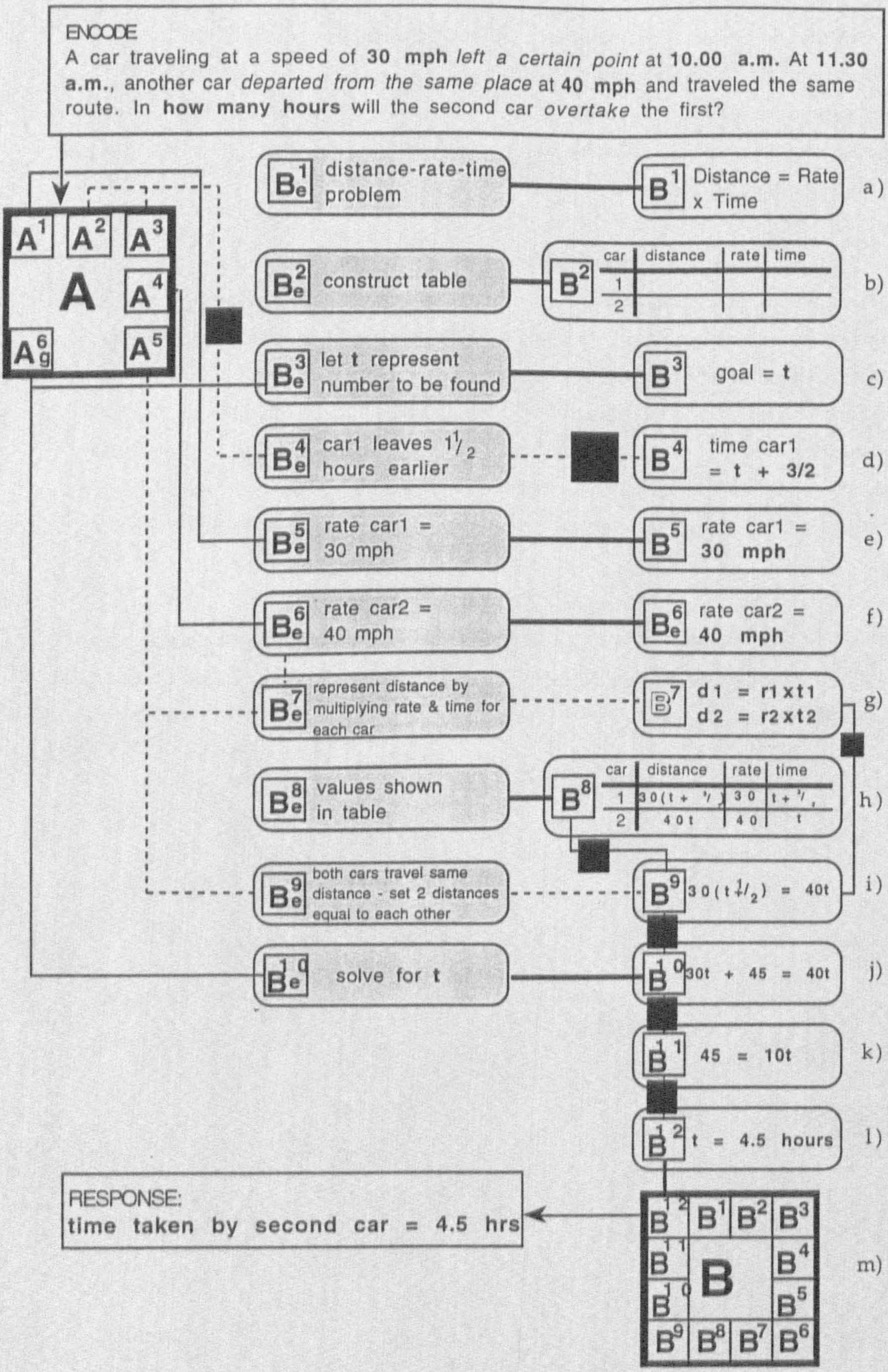


Figure 5.5. The practice problem.

3.1.2. Predicted sources of difficulty in mapping and adapting the practice problem to solve a close variant

In Reed et al.'s experiments subjects were presented with the following equivalent problem:

A car travels south at the rate of 30 mph. Two hours later, a second car leaves to overtake the first car, using the same route and going 45 mph. In how many hours will the second car overtake the first car?

Figure 5.6 shows the problem givens.

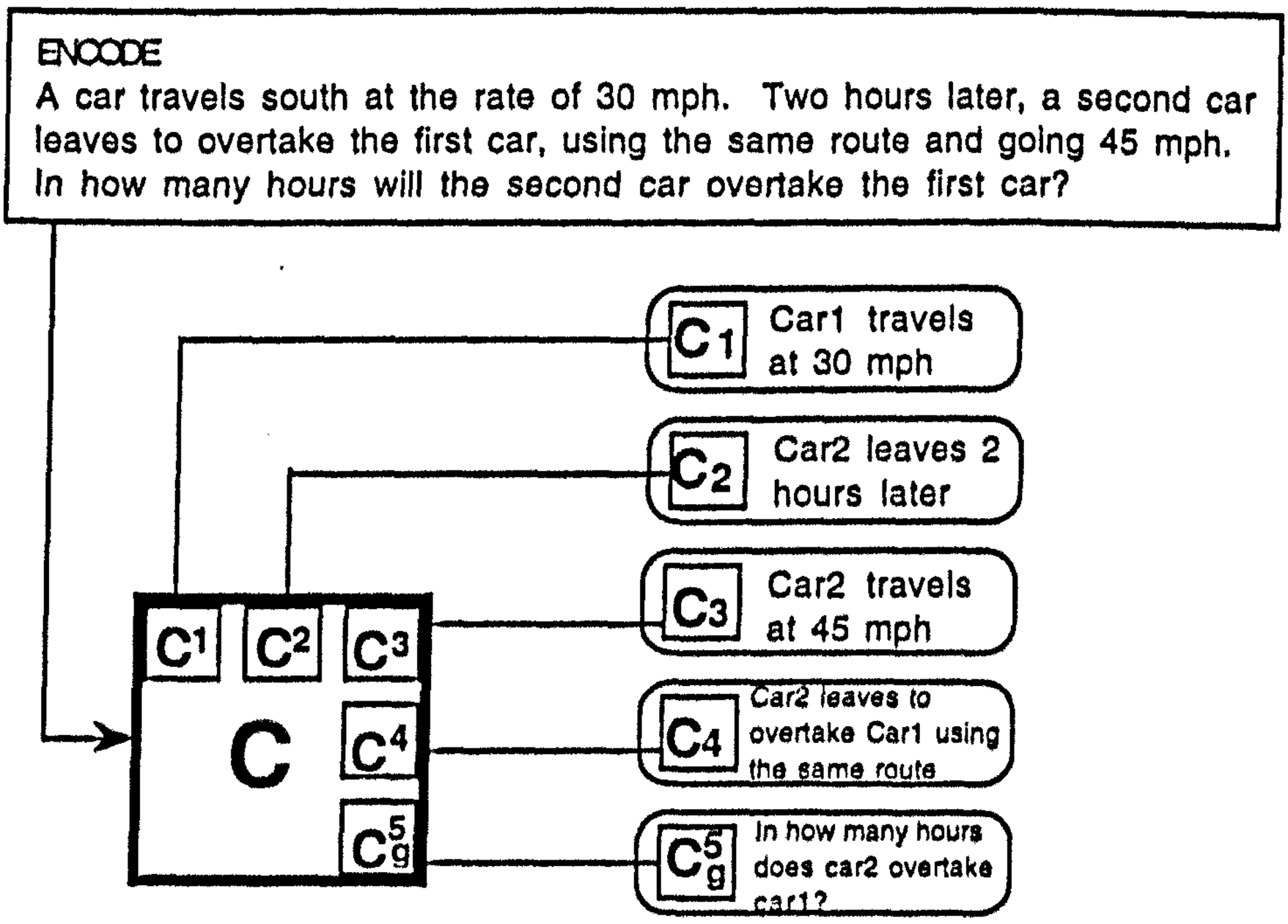


Figure 5.6. 'Equivalent' test problem givens.

As figure 5.7 shows, there is no direct match between the '2 hours' mentioned in the test problem and the times given in the training problem (represented by A^2 and A^3 respectively). For this reason the access line from C^2 is shown as being blocked. Subjects have to infer that they have to find the difference in departure times (represented by D^4). It is at this point that the 'equivalent' problems veer away from isomorphism and the mappings depicted in row *d* are therefore a source of potential error.

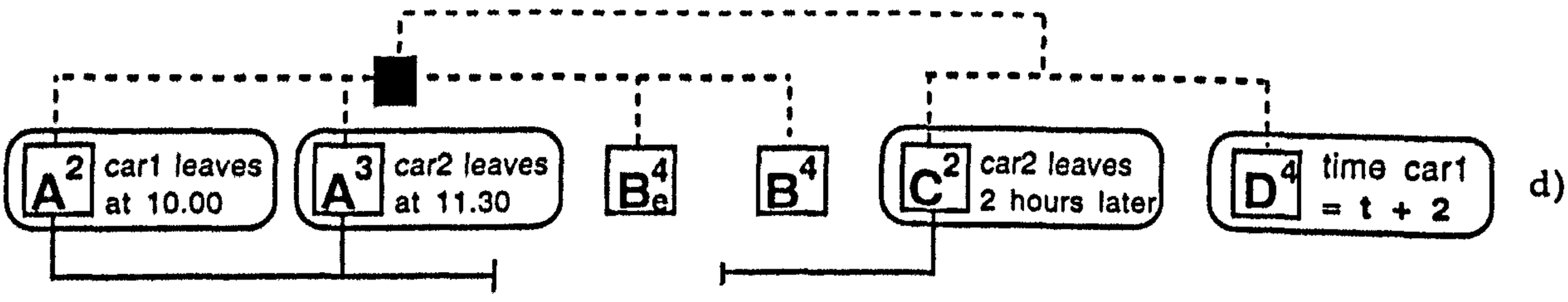


Figure 5.7 Mapping and adapting times in test and elaborated practice problems.

Some of the elaborated training problem givens can be mapped directly onto the equivalent test problem. The first car in both problems travels at 30mph. This information is represented as A^1 and C^1 in row *e* of figure 5.8. The speeds can be mapped onto the variables in the relevant equation (B^5 and D^5 in row *e*). By changing the rate of the second car in the training problem from 40mph to 45mph in the test problem (A^4 and C^3 in row *f*) the solver could also map this onto the rate term in the equation (D^6).

When using the example to solve an equivalent problem, the solver 'inherits' all the difficulties mentioned in understanding the example problem. Apart from the added difficulty of mapping the times in row *d*, subjects will have difficulties where there are dotted lines and black boxes in figure 5.8. Those difficulties have already been described and represented in figure 5.5. (A more detailed description of figure 5.8 can be found in section 5.2 of Chapter 4.)

The summary figure 5.8 demonstrates where mappings are blocked and where the solver is still obliged to make inferences about which operators should be used in order to relate the problem givens to the solution. We can therefore make further predictions about the possible sources of difficulty for the subjects:

Hypothesis 2: Any move away from isomorphism, however slight, will be a source of error. Some subjects will therefore have difficulty at line *d* since the times in the example do not map directly onto the one in the test problem.

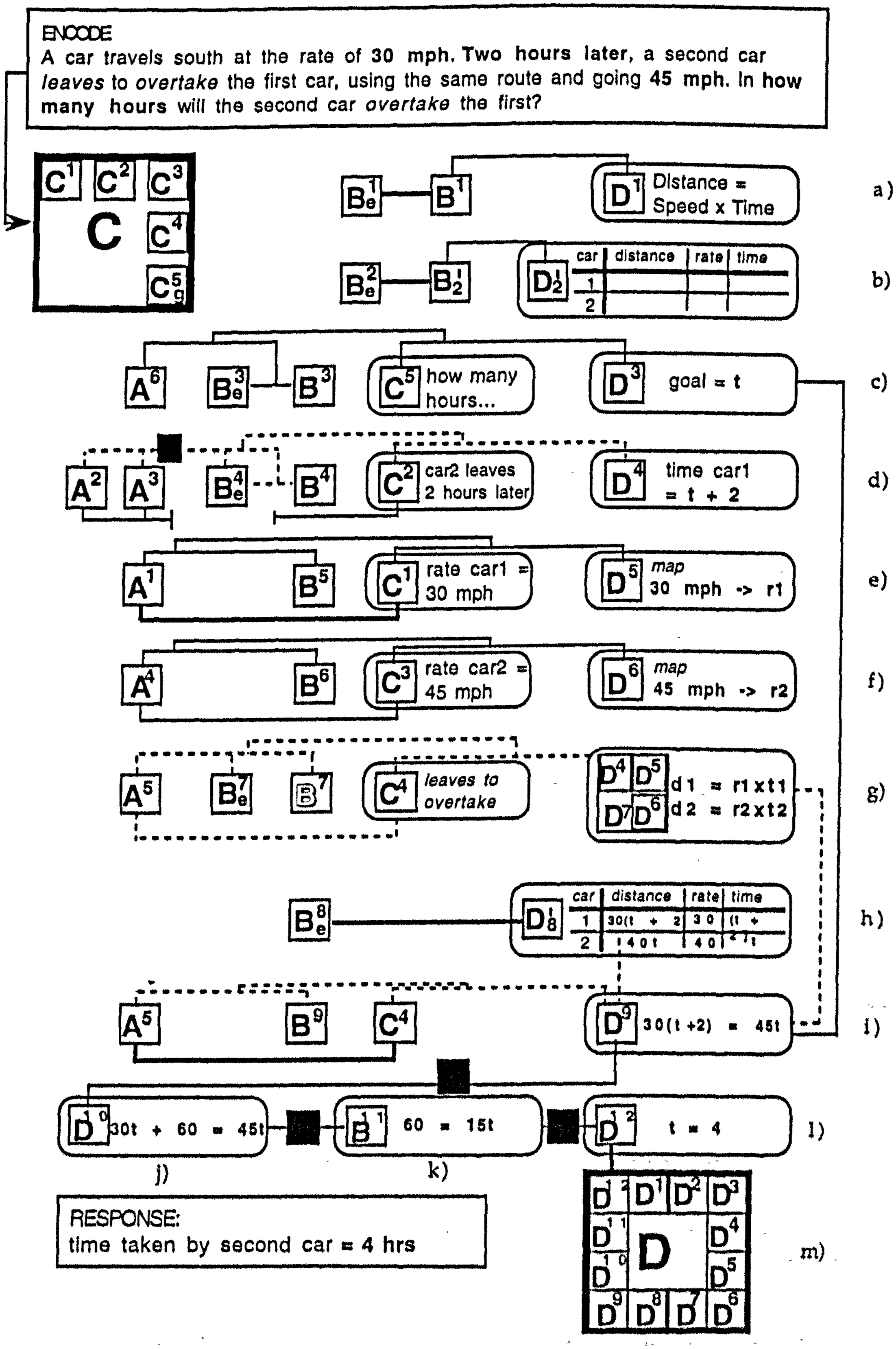


Figure 5.8. Mapping of a close transfer problem onto the training problem.

3.1.3. Predicted sources of difficulty in mapping and adapting the practice problem to solve a distant variant

Reed et al.'s 'similar' test problem was as follows:

> *A pick-up truck leaves 3 hr after a large delivery truck but overtakes it by travelling 15 mph faster. If it takes the pick-up truck 7 hr to reach the delivery truck, find the rate of each vehicle.*

The problem givens are shown in figure 5.9.

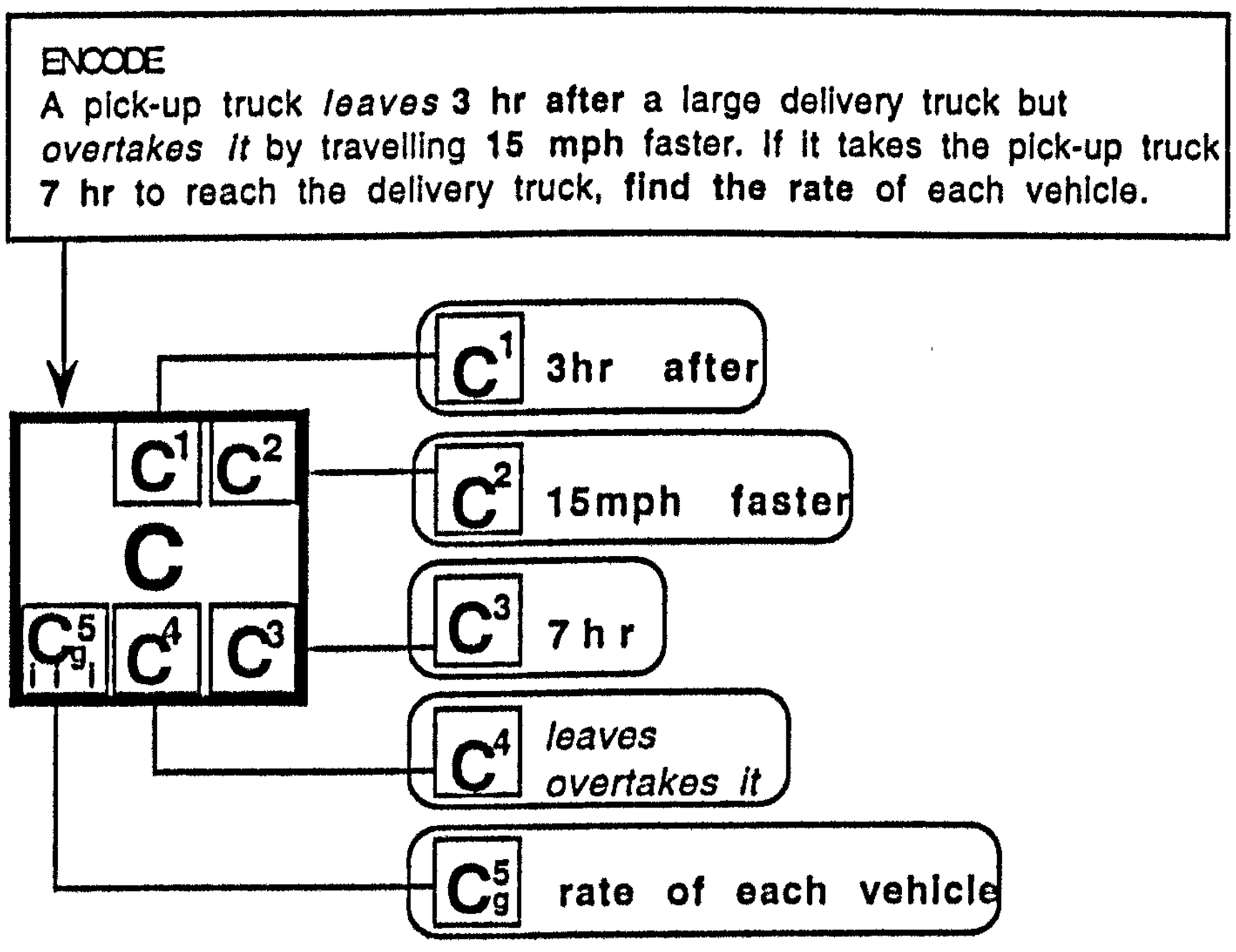


Figure 5.9. The problem givens for the distant variant of the example.

Mapping the practice problem onto a distant variant gives a quite different picture. None of the values in the problem statement (C^1, C^2 and C^3 in figure 5.9) map onto anything in the earlier problem.

In figure 5.10 in rows *c*, *d*, *e* and *f*, the lines from the C terms are shown as blocked. The goals of the two problems are different; the procedure for finding rates is different from that for finding the time taken by a vehicle. This will affect the subjects' ability to complete the table in row *h* and generate the equation in row *i*.

The summary figure 5.10 shows not only the inferences inherited from understanding the example problem (the A and B terms) but also specifies where subjects would have

difficulties making the necessary mappings. From the analysis of the problem we would predict that subjects will have difficulty at the following lines:

c) since the goals of the two problems are different;

d, e) since the problem elements do not map and the subject is not told what operator to apply;

f) nothing in the earlier problem maps onto a difference in rates and the subject is not told what operator to apply;

g) for the reasons already given in the discussion of the 'equivalent' problem subjects may have difficulty generating the two halves of the relevant equation;

h) the types of elements in the table in the training problem do not match those in the test problem (there are variables in the *time* column in the training problem but not in the test, and variables in the test problem under *rate* but not in the training problem);

i) subjects have to solve for r rather than t and then find the rate of the other vehicle in subgoal *m*.

In fact very little of the original example problem is of any use at all.

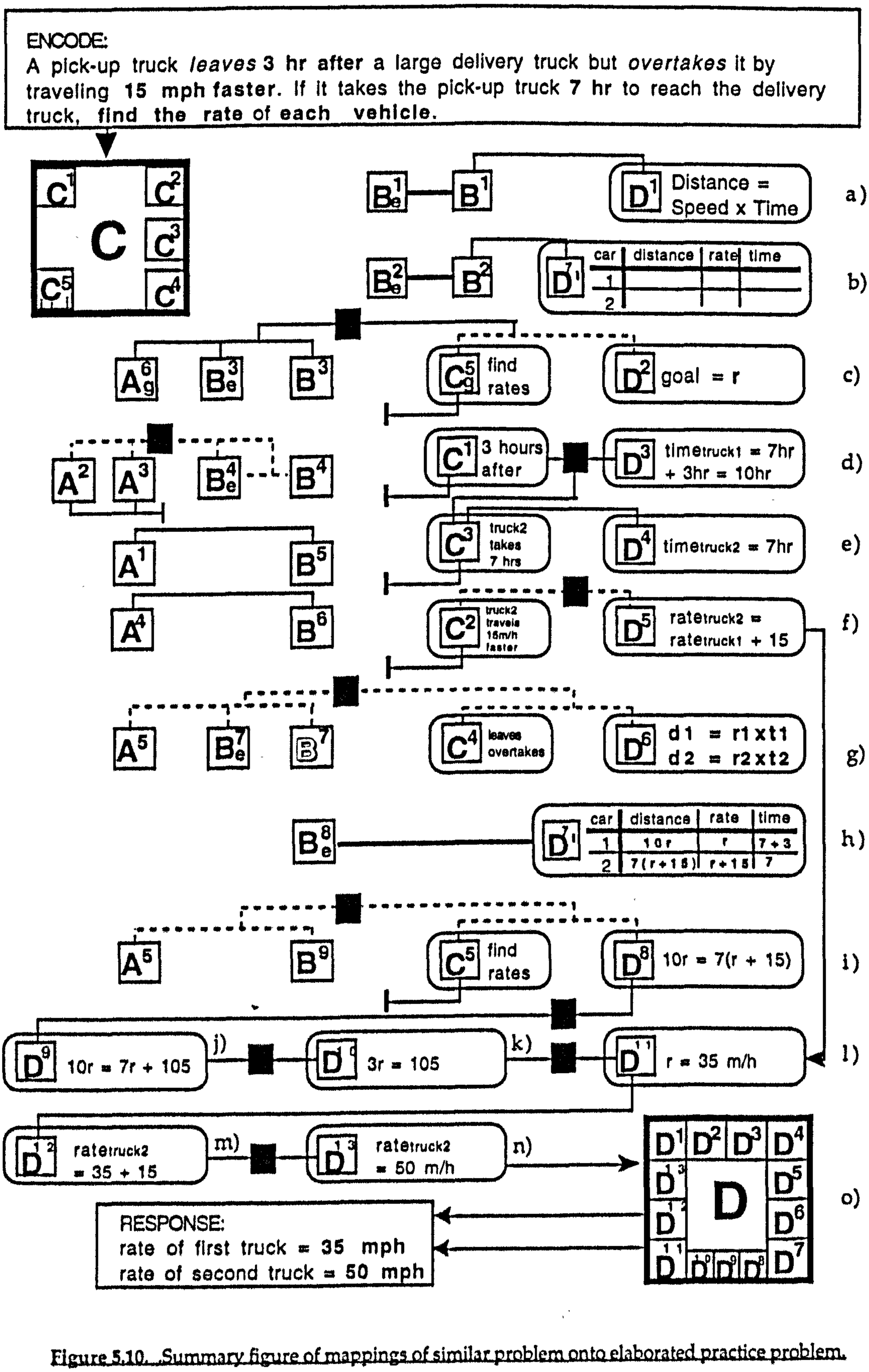


Figure 5.10. Summary figure of mappings of similar problem onto elaborated practice problem.

Because of the difficulty in mapping values across from source to target, and because there is no general procedure presented for adapting the example problem to this distant variant, we can make further predictions about the difficulties facing students attempting to imitate a distant variant:

Hypothesis 3: There will be more 'quantity errors' (errors due to misassigning values in the problem to variables in the equation) in the distant variant than in the close variant, since some values are not given in the exercise problem which are given in the example and vice versa.

Hypothesis 4: Related to hypothesis 3, we would expect that imitation would lead to 'over-transfer' from the source problem. In particular we would expect to find subjects attempting to use the variable t in the solution to the distant variant *because that is what it says to do in the example,* even though the value of t can be readily derived from the problem statement

One of the predictions from IPS is that subjects attempt to map values from a source to a target and that, when any manipulations are required of the underlying solution, this will create difficulties. In the case of these algebra problems subjects have to adapt the equation to solve the distant variant. Specifically, this means finding a difference in rates and adding that to the equation as well as solving for *rate* instead of *time*. This difficulty will be apparent in the number of subjects who are able to generate the relevant equation.

Hypothesis 5: There will be more 'frame errors' (errors occurring when attempting to generate and adapt the relevant equation) in the distant variant when no information is provided about how to adapt the equation.

It follows from the above analysis of the word problems that, if the inferences inherent in the Reed et al. explanation are removed and a more general procedure provided, then subjects will find it easier to map values across and adapt the procedure to more distant variants. Figure 5.11 represents the practice problem with the inferences removed. Row *a* represents the requirement to identify the problem as a *distance-rate-time* problem and provides the general overall equation governing such problems. Row *b* shows that this subtype involves 2 vehicles, so there are two *distance-rate-time* equations. Row *c* represents an explanation that the distances are the same for the two vehicles so the *rates x times* are equal. Rows *d*, *e*, and *f* show the mappings of values in the problem givens onto the equation. Row *g* represents the fact that the time for one vehicle can be stated in terms

of the time for the other, since we know the *differences* in times. This is mapped onto the equation in row *h*. The completed equation is presented in row *i*. No information is given about how the solution is derived from the equation (the point of the experiment was to see how well subjects could understand word problems well enough to generate the correct equation. Its purpose was not to explain the arithmetic operators necessary to solve algebra equations) .

When the example represented in figure 5.11 is presented as a *specific* case of a *general* procedure, it should be possible for solvers both to see how the general procedure operates in this example and to adapt it to fit a new problem of the same type. Hypothesis 6 is therefore:

Hypothesis 6: Removing inferences and providing a more general procedure will enhance transfer to distant variants of a problem type.

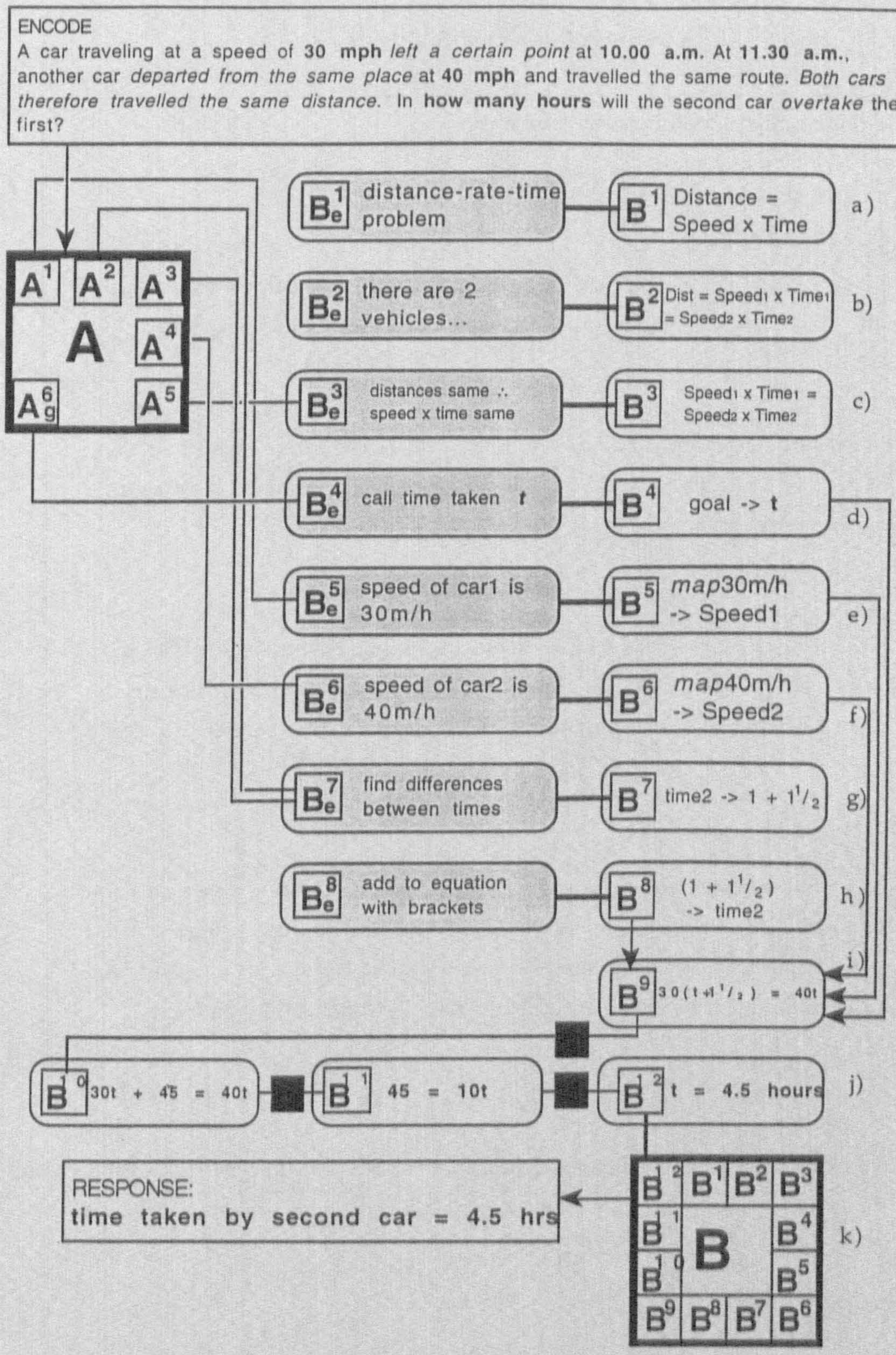


Figure 5.11. The practice distance problem with most inferences removed. This figure was used to generate the practice problem described in section 3.2.2.2.

For reference the hypotheses are repeated in table 5.1.

Table 5.1. Summary of experimental hypotheses

Hypothesis 1

The major difficulty in solving a close variant of the problem is one of mapping corresponding values. Where the transfer problem is literally similar, we would not expect mapping to pose a particular problem when they are imitating the source.

Hypothesis 2:

Any move away from isomorphism, however slight, will be a source of error. Some subjects will therefore have difficulty at line *d* since the times in the example do not map directly onto the one in the test problem.

Hypothesis 3:

There will be more 'quantity errors' (errors due to misassigning values in the problem to variables in the equation) in the distant variant than in the close variant, since some values are not given in the exercise problem which are given in the example and vice versa.

Hypothesis 4:

Related to hypothesis 3, we would expect that imitation would lead to 'over-transfer' from the source problem. In particular we would expect to find subjects attempting to use the variable t in the solution to the distant variant *because that is what it says to do in the example*, even though the value of t can be readily derived from the problem statement

Hypothesis 5:

There will be more 'frame errors' (errors occurring when attempting to generate and adapt the relevant equation) in the distant variant when no information is provided about how to adapt the equation.

Hypothesis 6:

Removing inferences and providing a more general procedure will enhance transfer to distant variants of a problem type.

3.2. Method

3.2.1. Subjects

The subjects were 40 Scottish secondary school students, aged between 14 and 16, who took part in the experiment during their normal mathematics classes. They were divided into four groups of 10, and randomly assigned to four conditions: Table/Close, Table/Distant, Mapping/Close and Mapping/Distant.

3.2.2. Materials

In the Table/Close and Table/Distant conditions subjects received a training problem along with the explanation used by Reed et al. which included a table as an intermediate representation of the solution procedure. The only difference was that the word 'rate' was replaced by 'speed'. In the Mapping/Close and Mapping /Distant conditions subjects received the same problems but the explanations were presented in more detail and the mappings between the elements of the problem and the equation were explicitly presented. The explanatory texts used are reproduced below:

3.2.2.1. Table Condition - Practice Distance Problem

A car travelling at a speed of 30 miles per hour (mph) left a certain place at 10.00 a.m. At 11.30 a.m. another car departed from the same place at 40 mph and travelled the same route. In how many hours will the second car overtake the first car?

The problem is a distance-speed-time problem in which

$$Distance = Speed \times Time$$

We begin by constructing a table to represent the distance, speed, and time for each of the two cars. We want to find how long the second car travels before it overtakes the first car. We let t represent the number that we want to find and enter it into the table. The first car then travels $t + {}^3/_2$ hr because it left $1{}^1/_2$ hrs earlier. The speeds are 30 mph for the first car and 40 mph for the second car. Notice that the first car must travel at a slower speed if the second car overtakes it. We can now represent the distance each car travels by multiplying the speed and the time for each car. These values are shown in the following table.

Car	Distance (miles)	Speed (mph)	Time (hrs)
First	$30(t + {}^3/_2)$	30	$t + {}^3/_2$
Second	40 x t	40	t

Because both cars have travelled the same distance when the second car overtakes the first, we set the two distances equal to each other:

$$30(t + {}^3/_2) = 40t$$

Solving for t yields the following:

$$30t + 45 = 40t;$$
$$45 = 10t;$$
$$t = 4.5\ hr.$$

3.2.2.2. *Mapping Condition - Practice Distance Problem*

The worked out problem shown below was presented to subjects in the Mapping Condition. To show that most of the inferences required to understand the explanation in the Table Condition have been removed, the letters in square brackets in the margin relate to the lettered rows in figure 5.11.

A car travelling at a speed of 30 miles per hour (m/h) left a certain place at 10.00 a.m. At 11.30 a.m. another car departed from the same place at 40 m/h and travelled the same route. Both cars therefore travelled the same distance. In how many hours will the second car overtake the first car?

As you can see, the two cars travel at different speeds and leave at different times:
The first car leaves $1^1/_2$ hours before the second car (11.30 - 10.00);
The second car travels 10 m/h faster than the first one (40 m/h - 30 m/h).

Since the first car has already been travelling for $1^1/_2$ hours you can work out the distance it travelled. If it goes 30 miles in an hour then it will have travelled 45 miles in $1^1/_2$ hours (30 m/h x $1^1/_2$ hours).

The problem now becomes: How long will it take the second car to make up this 45 miles by travelling 10 m/h faster?

In 1 hour the second car will travel 10 miles more than the first car, so it will take 4.5 hours to make up the 45 miles (45 h + 10 m/h). In other words it will take 4.5 hours for the second car to overtake the first car.

1. Another way of solving this type of *distance-speed-time* problem is to use an equation:

a] a)If you know the speed and time of a vehicle, then you can find the distance it travelled by using the equation:

Distance = Speed x Time

For example, if a car travels at 40 m/h (Speed) for 2 hours (Time) then it goes 80 miles (Distance - found by multiplying 40 by 2).

b] b) In problems like the one above there are *two* vehicles (let's call them *vehicle1* and *vehicle2*). The equation for each vehicle would be:

distance = speedvehicle1 x timevehicle1

and

distance = speedvehicle2 x timevehicle2

c] c) But the two vehicles travel the *same distance*. So the distance travelled by one vehicle is equal to the distance travelled by the other. If the *distances* are equal then the *speed x time* for each vehicle must be equal too, so the equation to use when there are *two* vehicles travelling the *same* distance is:

$$\text{speed}_{\text{vehicle1}} \times \text{time}_{\text{vehicle1}} = \text{speed}_{\text{vehicle2}} \times \text{time}_{\text{vehicle2}}$$

2. We can now fill in some of the information that the problem gives us into this equation.

d] The problem gives the speeds of both vehicles but not the time taken by either of
e] them. We want to find the time taken by vehicle2, so for the moment we will call it
f] *t*.

speedvehicle1 x timevehicle1 = speedvehicle2 x timevehicle2

3. The next thing to do is to look at any *differences* between the two vehicles which we can then use to fill in the blanks in the equation.

g] We know the two cars left at different times, so that the first car travels $1^1/_2$ hours longer than the second one. The second car takes 't' hours so the first car takes $t + 1^1/_2$ hrs. We can now add this to the equation:

30 $(t + 1^1/_2)$ 40 t

speedvehicle1 x timevehicle1 = speedvehicle2 x timevehicle2

You end up with:

i] **30 x $(t + 1^1/_2)$ = 40 x t**

Solving for t gives you the time taken by vehicle2 which is what you have to find.

30t + 45 = 40t

45 = 40t - 30t

45 = 10t

j] **t = 4.5 or $4^1/_2$ hours**

3.2.3. Design and Procedure

The experiment used a 2x2 between-subjects design. The independent variable was the type of instruction provided with the training problem. The subjects were asked to take part in a study of instructional materials and were told that the study was not a test of their own problem solving abilities. They were then presented with three sheets containing the instructions, the example problem and associated solution procedure, and the exercise problem. The instructions were as follows:

In a moment you will be asked to read the next sheet (sheet B) which gives an example problem and an explanation of how to solve it. Please read it carefully.

Once you have read sheet B, turn to the third sheet (sheet C) and write your name and class at the top. Then try to solve the problem which you find there.

Please use the example problem on sheet B to help you solve the problem on sheet C.

Feel free to make as many rough notes as you want on the sheet. In fact it would be helpful if you didn't rub or Tippex out any of your notes.

Please do your best, but don't worry if you find some of it too difficult.

When you are sure you have understood these instructions start reading sheet B.

There was no time limit imposed but the experiment was conducted within the normal lesson time of approximately 35 minutes. Subjects were allowed to consult the example solution throughout the experiment. At the end of the experiment subjects were also asked to write any comments they had about the usefulness or otherwise of the example.

3.3. Results

Novick and Holyoak's study identified three levels of performance which were scored in the following way: a fully correct solution (score of 2), a partial solution (score of 1), and no transfer (score of 0). The present study identified similar levels of response but also included identifying the correct elements that took part in the equation as a necessary prerequisite of constructing a correct equation. Responses were also analysed for the type of equation used, the degree to which subjects were able to map elements of the problem statement onto the relevant equation, and the types of errors committed.

3.3.1. Analysis of solutions.

Correct solution. The solution was coded as being correct if the correct equation was used and the correct procedure employed. A correct solution also contains the correct values in all variable slots in the equation. The correct procedure here is defined as the procedure given in the training example.

Correct equation. The correct equation means that the subject has filled in the values in the equation: $speed_{vehicle1} \times time_{vehicle1} = speed_{vehicle2} \times time_{vehicle2}$. Note that an equation can be classified as correct even though it may contain quantity or matching errors. This classification is in conformity with that used by Reed et al. (1985).

Correct answer. A correct answer can be generated without using an algebraic method. In a few cases the correct answer was given despite errors in the calculation and sometimes in

cases where it was difficult to ascertain exactly what procedure was used. (This was the case mainly in Experiment 3.)

Overall score. The overall score depends on whether: the subject used a correct solution (score of 3); the subject generated a correct equation as defined above (score of 2); the subject has identified all the elements necessary for the solution but has not mapped them onto the equation (score of 1); and no transfer (score of 0). This scoring scheme approximates that used by Novick and Holyoak (1991).

Note that the aim of the experiment was to identify how well subjects could generate the correct equation using the correct procedure. It did not concern itself with the subjects' ability to manipulate the equation to generate the correct answer once the equation had been generated.

3.3.2. Errors

Responses were also examined to identify the types of errors made. The errors were classified as follows:

Non-algebraic solution. In some cases subjects used both an algebraic and a non-algebraic solution method; in such cases the non-algebraic solution method was ignored for the purposes of coding. Otherwise subjects are coded as using a non-algebraic method whether or not the method gives the correct answer.

Wrong equation. Subjects are coded as using the wrong equation when it appears they are using any equation other than the 'correct' one as defined above.

Quantity errors. A quantity error refers to an error produced when the subject maps the wrong value from a target problem to the source. An example would be mapping the *difference* between the speeds of the vehicles (e.g. 15m/h) onto the *speed* of one of the cars in the source.

Matching errors. A matching error occurs when the subject substitutes a value in the source problem for a value in the target without any attempt at adapting it.

Frame errors. A frame error is an error in the form of the equation. *If* $30(t + 2) = 45t$ is the correct equation then an equation such as $30(t + 2) = 45$ would be a frame error since it does not preserve the form of the original *speed x time* equation (the *time* of the second vehicle is missing).

3.3.3. A comparison of success rates in the Table and Mapping Conditions

Figures 5.12 and 5.13 present the results of the four groups. The raw data are tabulated in Appendix 2.

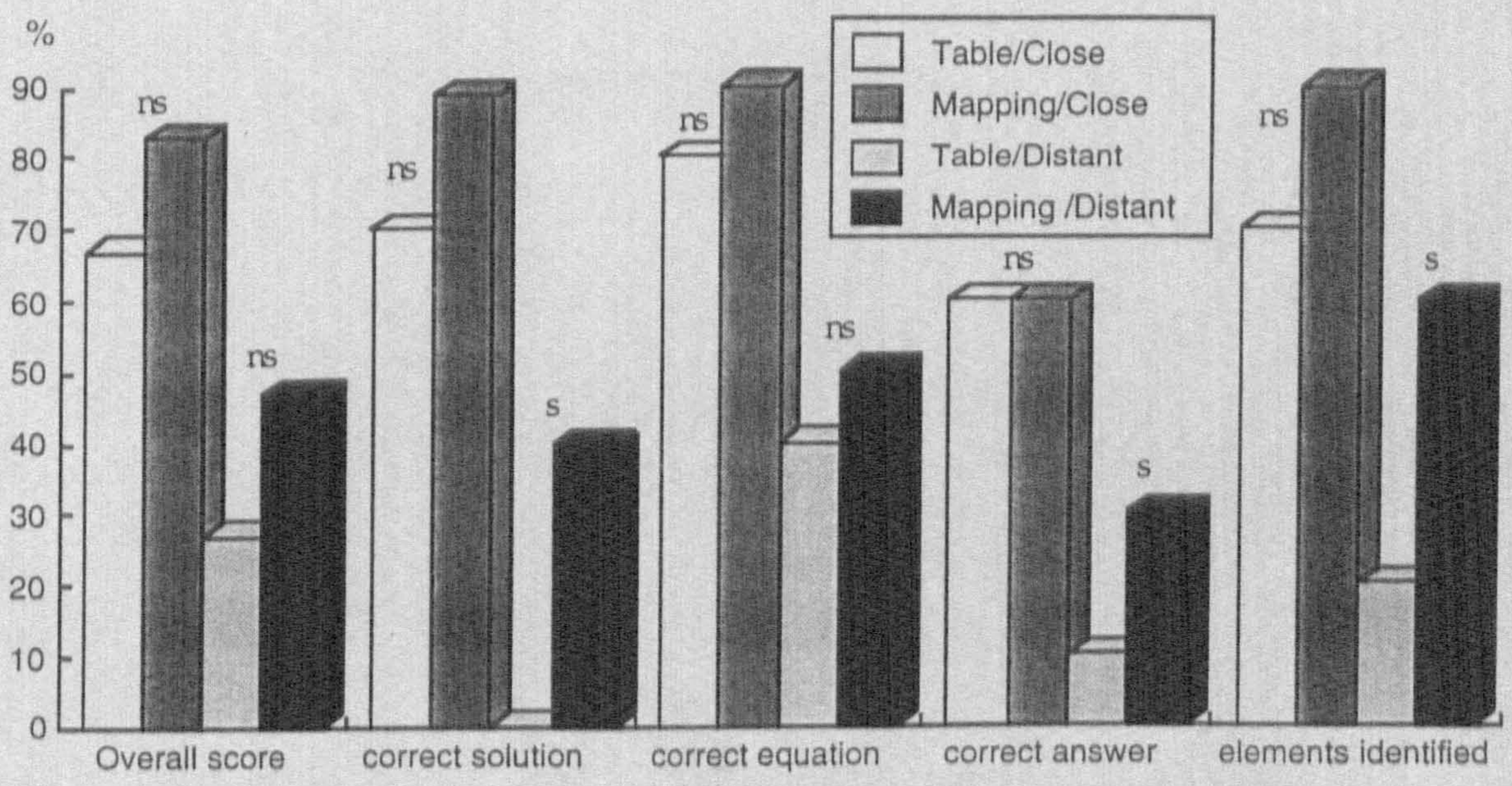


Figure 5.12. Success in the Table and Mapping Conditions. 'ns' refers to differences in adjacent columns which are not significant and 's' to those which are.

According to hypotheses 1 and 3 we would expect a greater difficulty in mapping and more quantity errors in the Distant conditions. There should therefore be fewer elements identified in those conditions. 70% of subjects in the Table/Close Condition correctly filled out the boxes in the table for *time* and *speed* for the two vehicles. In the Table/Distant Condition only 20% correctly filled in those boxes. This difference is significant ($z=-2.4543$, $p < .008$; the figures refer to a z-test for $n < 30$). In the Mapping/Close condition, 90% identified the correct elements as opposed to 60% in the Mapping/Distant Condition. This difference is not significant ($p > .05$).

All subjects in the Table/Close Condition who filled in the *time* and *speed* boxes in the table also correctly filled in the *distance* boxes, whereas only 1 subject in the Table/Distant condition managed to do so.

In line with hypothesis 6, that providing more information would enhance transfer in distant variants, significantly more subjects in the Mapping/Distant condition identified the correct elements than in the Table/Distant condition ($z = -2.012$, $p < .05$). Significantly more correct solutions were generated in the Mapping/Distant Condition than in the Table/Distant Condition ($z= -2.528$, $p < .006$). Although 4 subjects in the

Table/Distant Condition had used the correct equation, 3 had the wrong values in the equation and the 4th had confused the trucks. On the other hand, the subjects in the Mapping/Distant condition got the correct answer by putting the correct values in the equation and following the correct procedure.

3.3.4. A Comparison of Errors in the Table and Mapping Conditions

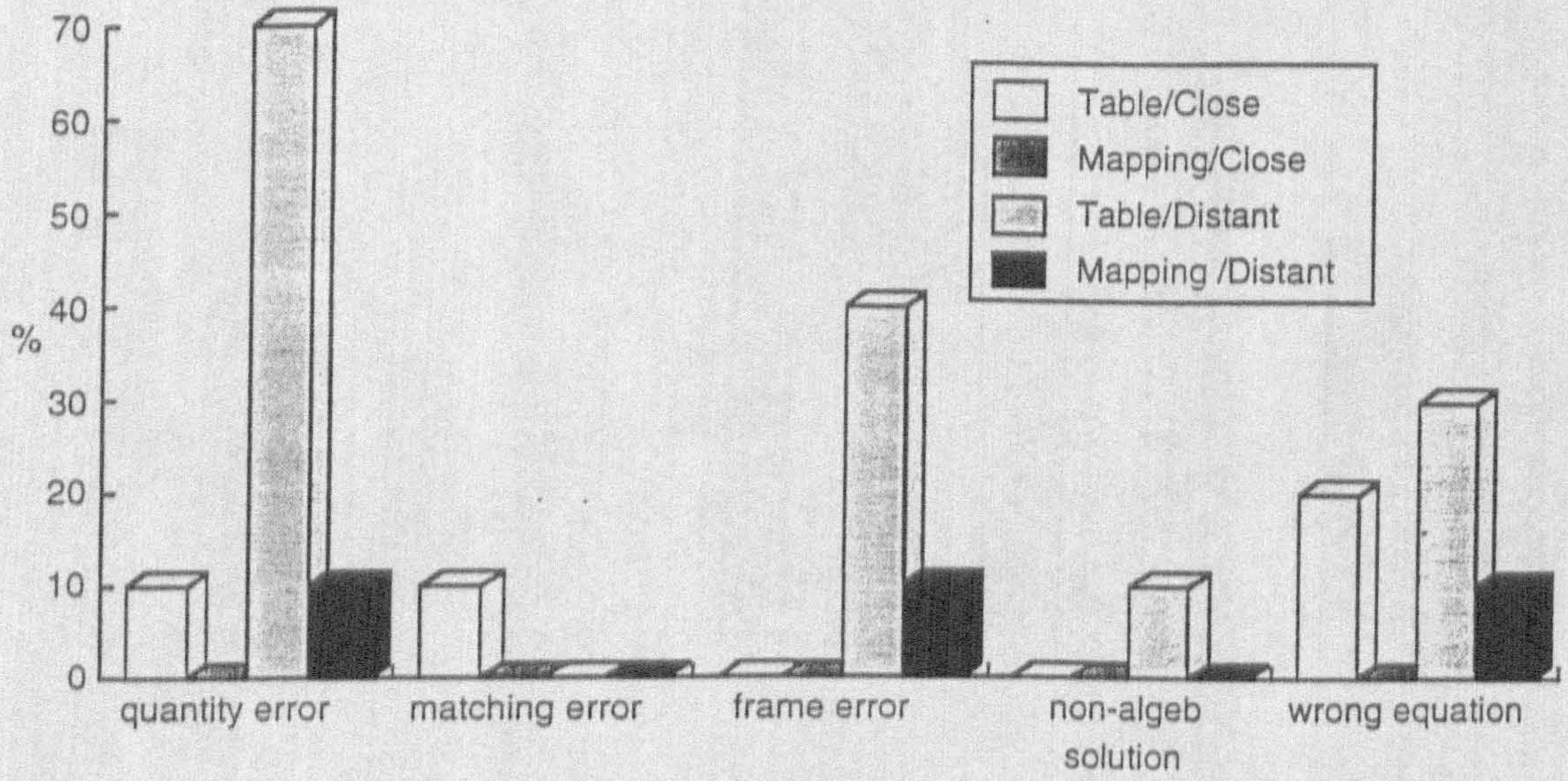


Figure 5.13. Errors in Table and Mapping Conditions.

Hypothesis 3 predicted increases in quantity errors in the distant variant of the example problem. Subjects made significantly more quantity errors in the Table/Distant Condition compared with the Table/Close Condition ($z = -3.018$, $p < .002$). In line with hypothesis 5, which predicted more frame errors when subjects were required to adapt the equation, a greater number of subjects generated the correct equation in the Mapping/Close condition than in the Table/Close condition ($z = -1.7833$, $p < .05$), but there was no significant difference in the Distant conditions. In the Table/Close Condition there were no frame errors, whereas in the Table/Distant Condition there were 4 such errors. This difference is significant ($z = -2.528$, $p < .006$). In the Mapping/Distant condition only one frame error and one quantity error were made.

Hypothesis 6 accounts for the fact that fewer errors were made in the Mapping Conditions than in the Table Conditions, since more information is provided about mappings and how to generalize the procedure. More wrong equations were generated in the Table/Close Condition than in the Mapping/Close Condition ($z = 1.7$, $p < .05$). Subjects in the Table Conditions attempted to use the *distance = rate x time* equation rather than the *rate1 x*

time1 = *rate2* x *time2* equation. In the distant variants fewer quantity errors were made by the Mapping group than by the Table group ($z = -3.02$, $p < .002$), as well as fewer frame errors ($z = -1.74$, $p < .05$).

Subjects' written comments provided some indication about where they found difficulty in understanding the training problem. In the Table/Close Condition most subjects successfully solved the problem, nevertheless they did not always find the explanation useful. Here are some of their comments:

S4: 'The distance in the table was quite confusing to start with.'

S7: 'explain clearer' (written against $t + 3/2$ in the instruction sheet).

S10: 'I don't really understand where you got the 45 from on the previous page [example problem]. I don't think (for an example) you have explained that well enough.' (This refers to the 45 generated from multiplying 30 by 3/2.)

3.3.5. 'Overt' imitation and over-transfer

In some cases subjects attempted to adapt the exercise problem in order to make it *perceptually* similar to the earlier problem. Such 'overt' imitation can take various forms (which are not always readily quantifiable). Examples would be: ensuring that the solution to the exercise problem has the same number of '=' signs or same number of lines as the example when solving for t. Overt imitation is a form of over-transfer, where the solver transfers too much information from the source.

Matching error is the most overt form of imitation where the subject copies a value from the source problem across to the target without any attempt at adapting it. S8 copied the $^3/_2$ from the example into the exercise problem. (She nevertheless managed to generate the correct answer. She wrote '$30(t + {}^3/_2) = 45t$' and then on the next line '$30t + 60 = 45t$.' It is difficult to know whether she simply made a slip of the pen when filling in the table by writing $^3/_2$ instead of 2; whether she made an arithmetic error when multiplying the $^3/_2$ by *30*; or whether she copied the $^3/_2$ since that is what was written in the source example, but multiplied 2 by *30* instead because it seemed to make more sense to do so.)

Five subjects in Table/Close condition used overt imitation. They adapted the solution so that it was perceptually similar to the source example. In doing so they converted the 2 hours difference between the departure times of the cars in the exercise problem to $^4/_2$ so that it looked the same as the example problem and yet still yielded the correct answer. Four of these subjects generated the correct solution (correct equation and solution

procedure). S7 provided an example of the mapping in his rough notes when he wrote: '$^3/_2$ = $1^1/_2$' and '$^4/_2 = 2$'.

Three subjects in the Table/Distant Condition used overt imitation, converting the 3 to 6/2 and the 7 to 7/2. In the latter case the conversion led to incorrect answers. S2 copied the table from the example with the same figures in the boxes then erased them. The subject did not, however, replace them with any of the values in the target.

Some subjects in the Table/Distant Condition continued to use *t* as a variable (as in the example) in place of the times of the vehicles even though the times were given or could be inferred (by adding 7 and 3). In this case the subject is copying the earlier procedure without adapting it. Reed and Ettinger found the same effect in their studies, where their subjects copied over structures from the example problem which were unnecessary in the target problem. Such 'structure matching errors' are a form of over-transfer and played a great part in frustrating their attempts to induce transfer in their subjects.

3.3.6. Difficulties in mapping.

There was some confusion in both Distant conditions in assigning vehicle labels (vehicle1, vehicle2) to the correct vehicles. The pickup truck in the 'distant' problem is mentioned first, even though it left after the delivery truck, and was therefore assigned to the vehicle1 role by two subjects in the Table Condition and two in the Mapping Condition. This misassignment led to quantity errors in the equation. In figure 5.5 the order in which the example problem givens are mentioned in the text is preserved in the A term box. That is, A^1 refers to the speed of the first car, which is the first of the problem givens mentioned in the text. Similarly, the order in which the exercise problem givens are presented is preserved in the C term box in figure 5.8. In this case, however, C^1 refers to the second vehicle since it is the first one mentioned. The order in which the givens are presented in the two problems is not the same. The misassignment of vehicle labels should therefore have been predicted.

Two subjects in the Mapping/Distant Condition confused the trucks. However, in one case the subject attached all the correct values to the vehicles. In other words the vehicle *labels* were irrelevant and did not affect the outcome.

Three subjects tried to map the values in the table to the wrong equation, either *distance = speed x time* or *speed = distance/time*. This difficulty may have arisen from two possible sources. The subjects may have attempted to use the speed equations which they already knew (some wrote the *distance-speed-time* triangle:

which they had previously learned). Alternatively they may have used the *distance = speed x time* equation given in the Table conditions' explanations. In the Table/Distant Condition, one subject wrote:

S18: 'I don't understand how they work out the time when they don't have the distance.'

Clearly this subject did not understand the relation between the *distance* and *speed x time* of the vehicles and the fact that both travel the same distance.

In the Mapping conditions some attempted the non-algebraic solution as well as the algebraic one, none was successful at finding the correct answer from the former method (although some were successful at the latter). Some subjects attempted the non-algebraic solution, got confused and gave up. S39, for example commented:

'Too hard. I don't like the way in which it is written out. The example is too long and too hard to understand.'

Some, such as S23 and S28, found the non-algebraic solution unnecessary:

S23: 'This was easy to work out, once you knew the formula, but all the writing beforehand was confusing.'

S28: 'It was a very good and full explanation but some unnecessary notes.'

S40: 'This explanation goes into too much detail and unless you always do it that way it is difficult.'

Filling in the table proved very difficult for almost all subjects in the Table/Distant Condition, as Reed and Ettinger also found. Only one subject correctly filled out the Distance column suggesting that the relationship between the Speed and Time boxes and the Distance boxes was not made clear. Four subjects out of 10 in the Table/Distant Condition generated the correct equation; only 2 correctly filled in the table, and of them only one filled in the *distance* boxes. These figures suggest that the relation between the table and the equation was likewise not sufficiently explained. This should not be

surprising given that the relevant equation was not specified in the text. This result is contrary to that of Reed and Ettinger, who found that the equation could be readily generated once the values in the table were known.

3.4. Discussion

It was predicted in hypothesis 1 that solving a close variant of an example problem by imitation would not cause significant difficulties. 90% of subjects in the Mapping group and 70% in the Table group successfully solved the 'equivalent' problem.

A crucial aspect of the interpretation theory is its ability to specify exactly where subjects would go wrong when trying to imitate an example problem. In the Table/Close Condition, represented in figure 5.8, two potential sources of error were identified. The first was in line *d* of the figure where the times in the source and target did not correspond; the second was in generating the correct equation. According to hypothesis 2, if students are imitating a source, then variations, however slight, may lead to error. The only source of variation in the close variant was in generating the difference in times between two vehicles. In the example problem, this difference had to be computed. In the close variant, it was given. One might reasonably expect that the close variant should therefore be easy to solve since it involves one fewer subgoal. However, if subjects were closely imitating the source then that is precisely where any difficulties would occur. Of the 3 subjects who failed to solve the problem in the Table/Close Condition, two of them copied across $t+^3/_2$ from the example (this 'over-transfer' was predicted in hypothesis 4), and the third got as far as identifying both speeds but could not identify the times of the vehicles and failed to get beyond that point. Thus the analysis of the problem correctly identified the slight variation as being a source of potential difficulty.

There were a large number of quantity errors in the Table/Distant condition compared to the Close/Distant condition, as predicted in hypothesis 3. Subjects had difficulty mapping corresponding elements from one problem to another. This is because they were relying on the surface features of the problems to identify the mappings, rather than being guided by a principled understanding of the structure of the problems.

In line with hypothesis 4, examples of over-transfer were found in all conditions. In the distant conditions, subjects attempted to incorporate 't' into the equation since that was the variable used in the training problem. This was despite the fact that the times for the vehicles were given in the problem statement. In the close conditions, over-transfer was apparent in subjects who converted '2' to '$^4/_2$' or used the $t+^3/_2$ in the equation without any modification.

There were significantly more frame errors in Table/Distant condition compared to the Table/Close condition. Subjects were unable to adapt the equation to solve a distant variant. Similarly, some subjects were unable to identify the correct equation in the first place. In particular they tended to use distance = rate x time or a variation of it, such as rate = distance / time. The latter was ususally generated by the subjects themselves from the distance-rate-time triangle.

There was a significant increase in the number of quantity and frame errors in the distant variant, as predicted by hypotheses 3 and 5, but only in the Table conditions. Hypothesis 6 predicts these results since more information was provided in the Mapping Conditions about how the problem givens should be mapped to the underlying equation, and how this relates to other problems of the same type. It also led to a significantly greater number of correct solutions in the Mapping/Distant Condition than in the Table/Distant Condition.

It would appear, then, that the earlier analysis of the problems and of the mappings between them correctly predicted the areas where subjects would have difficulty. It also shows that attempting to improve explanations in areas of potential difficulty, by removing the need for inferencing and presenting a more general procedure, produces a greater degree of transfer to distant variants of a problem type.

4. Enhancing transfer and problem understanding: Experiment 3

In the Mapping/Distant Condition in experiment 2, 30% of subjects attempted a non-algebraic solution based on the non-algebraic solution in the example problem. None was successful, despite finding a correct solution using the algebraic procedure. Experiment 1 showed that subjects claimed to 'understand' example problems but were unable to adapt them to solve a new problem of the same type. It would be interesting to discover whether providing information about how to adapt the problem would produce a correlation between what subjects claim to understand and what they do understand based on their performance on a transfer problem.

For these reasons a second experiment was carried out to investigate the effects of removing the non-algebraic solution, which caused some confusion in the second experiment; and the effects of providing information about how to adapt a problem on subjects' assessment of their own understanding. In the present experiment, students were asked to rate their understanding of the two types of explanation given. The hypothesis was that students' ratings for the two problems would differ since the Table explanation involved more inferences than the Mapping explanation. As a consequence they should find the Table explanation more difficult to understand than the Mapping explanation.

This experiment was a replication of the Table/Distant and Mapping/Distant conditions of experiment 2 with a different sample of subjects and a simplified explanation in the Mapping condition. The hypothesis was therefore the same as hypothesis 6 of experiment 2, that removing inferences and providing a more general procedure would enhance transfer to distant variants of a problem type. The subjects in the present study had had at least two years more experience in algebra than those in experiment 2. Given their greater expertise it was expected that they would be better able to adapt the procedure in both the Table and Mapping Conditions.

4.1. Method

4.1.1. Subjects.

Subjects were 23 6th form students at a Secondary Comprehensive school in Milton Keynes. They had been successful in the GCSE exams in mathematics and were preparing for A-levels in mathematics.

4.1.2 Materials

The materials were similar to those used in Experiment 2 except that the mapping explanation was not so elaborated. For example, there was no explanation of the statement that it was 'a distance-speed-time problem in which distance = speed x time.' This is because the subjects were well acquainted with this general problem type (but not with colinear problems). Both the Table and Mapping explanations contained boxes on the right of the sheet so that students could note down their estimate of how well they understood the explanations provided by giving a rating as in experiment 1 (see below for details) . The Mapping explanation was as follows:

A car travelling at a speed of 30 miles per hour (m/h) left a certain place at 10.00 a.m. At 11.30 a.m. another car departed from the same place at 40 m/h and travelled the same route. Both cars therefore travelled the same distance. In how many hours will the second car overtake the first car? ☐

1. This is a *distance-speed-time* problem in which

distance = speed x time ☐

2. a) In this problem there are *2* vehicles so:

$\text{distance} = \text{speed}_{\text{vehicle1}} \times \text{time}_{\text{vehicle1}}$

and

$\text{distance} = \text{speed}_{\text{vehicle2}} \times \text{time}_{\text{vehicle2}}$ ☐

b) The two vehicles travel the *same distance*. If the *distances* are equal the *speed x time* for each vehicle must be equal too, so the equation to use when there are *two* vehicles travelling the *same* distance is: ☐

$\text{speed}_{\text{vehicle1}} \times \text{time}_{\text{vehicle1}} = \text{speed}_{\text{vehicle2}} \times \text{time}_{\text{vehicle2}}$ ☐

3. a) We can now fill in *some* of the information from the problem into this equation. We want to find the *time* taken by vehicle2, so for the moment we will call it *t*. (If you want to find the *speed* call it *s*.)

30 m/h ↓ 40 m/h ↓ t ↓

$\text{speed}_{\text{vehicle1}} \times \text{time}_{\text{vehicle1}} = \text{speed}_{\text{vehicle2}} \times \text{time}_{\text{vehicle2}}$ □

4. a) Next find any *differences* between the two vehicles and put them in the equation.

b) The cars travel at *different speeds* and leave at *different times*: All that's missing here is the $\text{time}_{\text{vehicle1}}$. The second car takes 't' hours, so the first car takes $t + 1^1/_2$ hrs (since it has been travelling for $1^1/_2$ longer). Add this to the equation with brackets round it: □

30 $(t + 1^1/_2)$ 40 t

↓ ↓ ↓ ↓

$\text{speedvehicle1} \times \text{timevehicle1} = \text{speedvehicle2} \times \text{timevehicle2}$

You end up with:

$30 \times (t + 1^1/_2) = 40 \times t$ □

Solving for t gives you the time taken by vehicle2 which is what you have to find.

$30t + 45 = 40t$

$45 = 40t - 30t$

$45 = 10t$

$t = 4.5$ or $4^1/_2$ hours □

4.1.3. Design and Procedure

There were only two conditions in this experiment corresponding to the Table/Distant and Mapping/Distant Conditions of Experiment 2. Subjects were given the following instructions:

> In a moment you will be asked to read the next sheet (sheet B, double sided) which gives an example problem and an explanation of how to solve it. Please read it carefully.
>
> You will see boxes on the right of the sheet. As you read the explanation on the sheet I would like you to fill in these boxes with the numbers 1 - 5 in the following way:
>
> 1. I don't understand this at all;
> 2. I don't think I understand this;
> 3. I think I have a rough idea what it's talking about;
> 4. I think I understand this reasonably well;
> 5. I understand this perfectly.
>
> Once you have read sheet B and filled in the boxes, turn to the third sheet (sheet C) and write your name and class at the top. Then try to solve the problem which you find there.
>
> **Please use the example problem to help you solve it.**
>
> Feel free to make as many rough notes as you want on the sheet. In fact it would be helpful if you didn't rub or Tippex out any of your notes.
>
> When you have finished please write any comments you may have about the way the problem is explained.
>
> Please do your best, but don't worry if you find some of it too difficult.
>
> When you are sure you have understood these instructions start reading sheet B.

The subjects were shown some examples on an overhead projector of how to fill in the boxes and asked if they had any questions.

4.2. Results

The results are shown in figures 5.14 and 5.15. (The complete data are tabulated in Appendix 2).

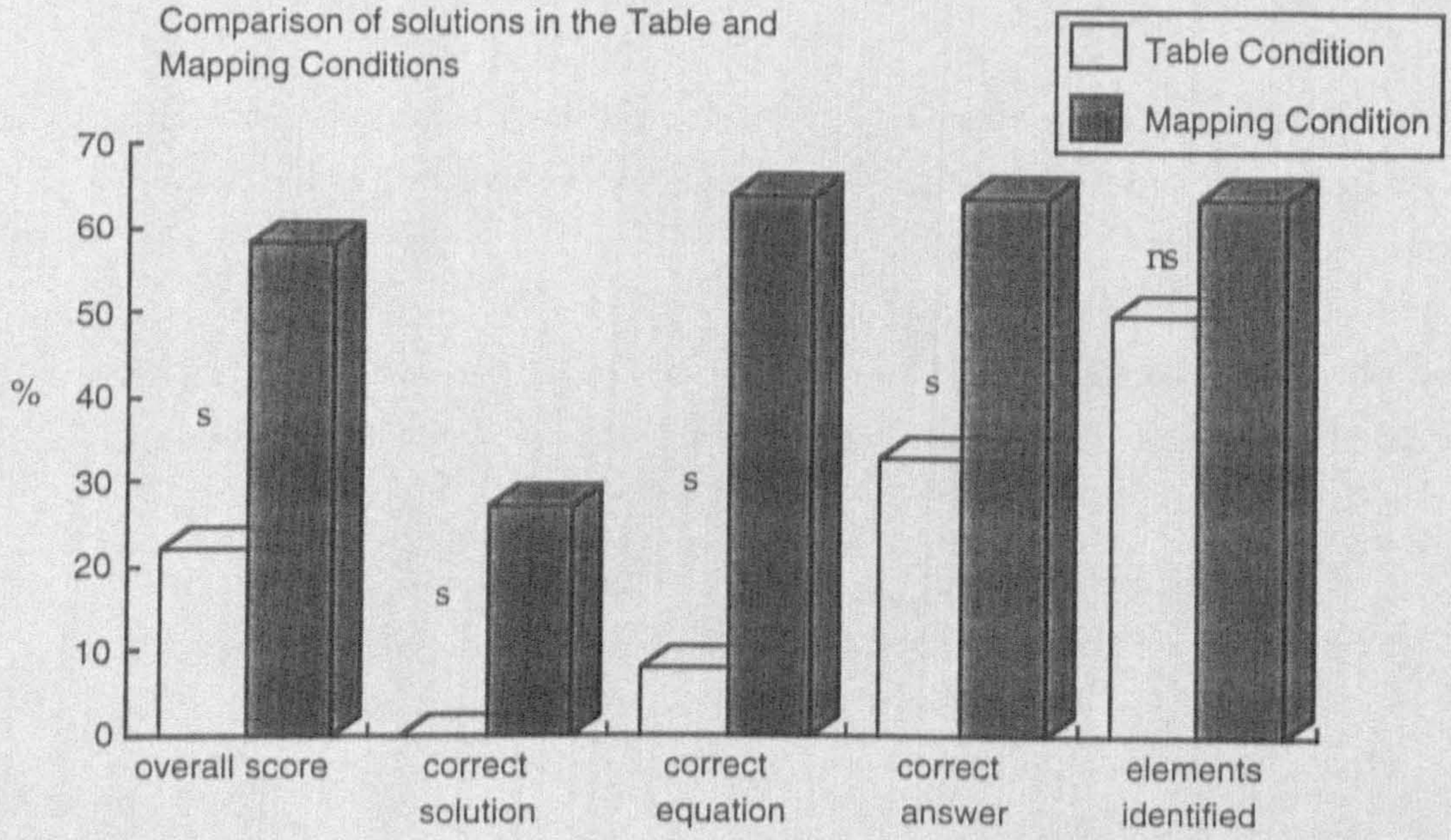


Figure 5.14. Success in the Table and Mapping Conditions (experiment 3).

As predicted, the Mapping Condition produced more correct solutions (where subjects used the correct equation with the correct values and the same procedure as in the example) than the Table Condition ($z = -1.94$, $p < .05$). Subjects in the Mapping condition were better able to generate the correct equation (the correct form of the equation, which may include wrong values) than those in the Table Condition ($z = -2.025$, $p < .05$). Subjects in the Mapping Condition were no better able than subjects in the Table Condition to identify or derive the values from the problem statement that mapped onto the variables in the equation. However, those in the Table Condition were unable to construct an equation out of those values. These differences can be seen in figure 5.14 by comparing the heights for the Table Condition of the 'correct equation' and 'elements identified' columns. This contrasts with the Mapping Condition where there was a significantly greater number of correct mappings of elements from the problem onto the equation ($z = -2.640$, $p < .005$). If the subjects in the Mapping Condition could identify the relevant elements, then they could readily derive the correct equation.

The overall score is useful as it can be taken as a summary of all the measures taken. These scores were significantly higher in the Mapping than in the Table Condition. The differences between scores were in the predicted direction (Mann-Whitney $U = 29$; $p < .01$, one-tailed).

In contrast with experiment 2, there was no significant difference in the number of quantity errors made in the two conditions (50% of subjects in the Table Condition and 33% in the Mapping Condition - see figure 5.15). However, there was a significant difference in the number of frame errors made ($z = -3.037$, $p < .002$). There was also a significant difference in the number of subjects in the Table Condition who accessed the wrong equation; 67% used an equation other than the relevant one ($z = -3.353$, $p < .0005$); none in the Mapping Condition did so. This also contrasts with the results from experiment 2 which showed no such difference.

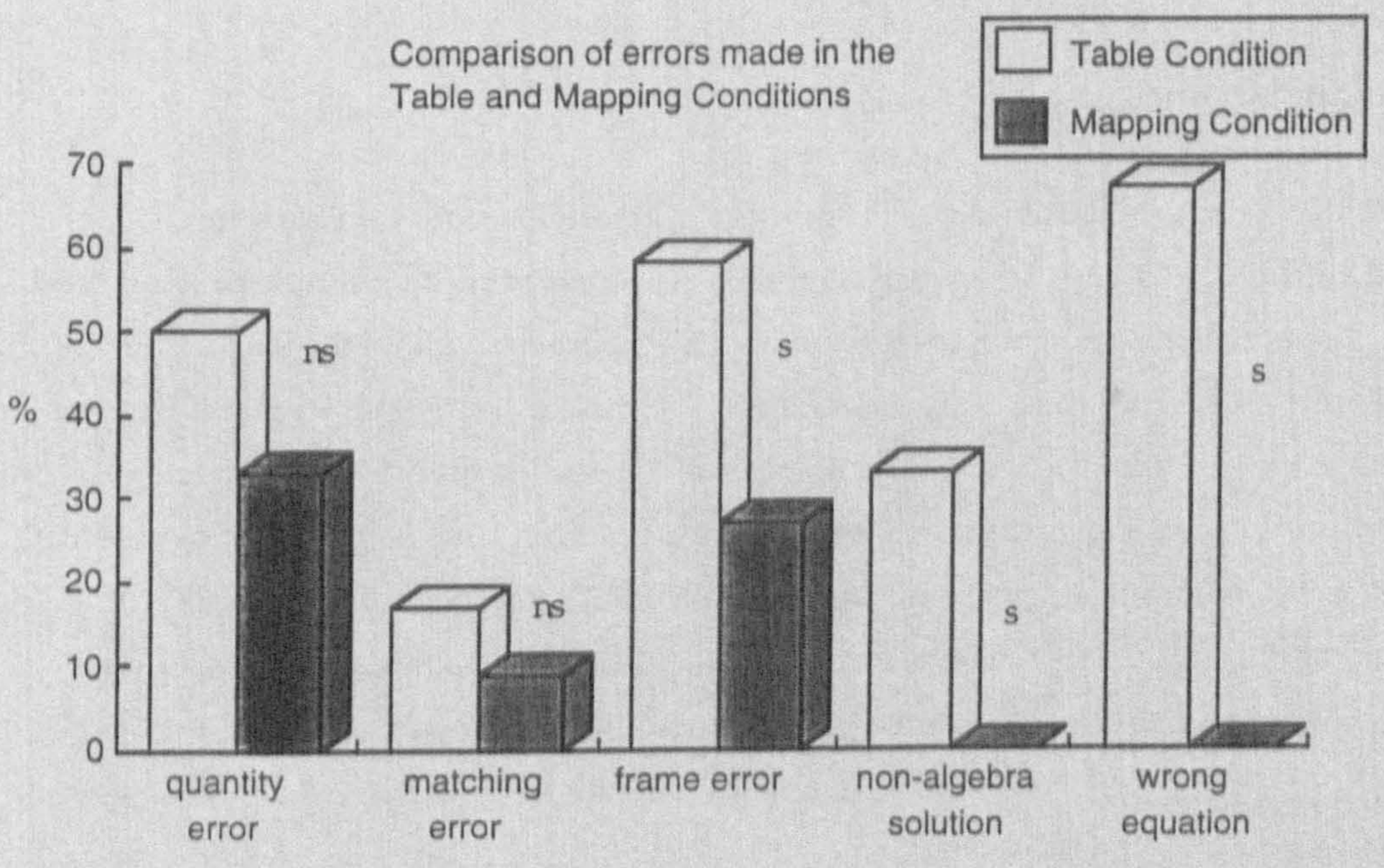


Figure 5.15. Types of error in the Table and Mapping Conditions (experiment 3). Significant differences are marked with an 's' and non-significant differences are marked 'ns'.

4.2.1. Subjects' comments

Subjects in the Table Condition reported that they found the example relatively easy to follow but couldn't see how it applied to the exercise problem. Typical comments were:

S3: 'I understand the example but when it came to the question I got a bit confused rearranging the formula myself';

S4: 'I understood how to rearrange the formula to get the speed but I could not continue as I did not have the distance travelled. I don't understand how to finish this. I did understand the question but there wasn't enough information.'

S5: 'I understand the explanation on [the example sheet] however I am unsure of how to solve the question above. Maybe more explanation was needed.'

As in experiment 2 subjects commented that they didn't see how to solve the problem when the Distance was not mentioned.

Comments on the effectiveness of the Mapping explanation were mixed. Subject S1 commented: 'Having read both examples I found this was the better of the two. It made substituting into the formula much more clear.'

The discrepancy between the subjects' own assessment of their understanding of the example and their actual performance is dealt with in section 5 of this chapter.

4.3. Discussion

The better performance of subjects in the Mapping Condition once again indicates the beneficial effect of removing inferences when subjects are reading an example problem, and of providing an explanation at a level of generality sufficient to allow the subjects to adapt the procedure to suit the target problem. The results clearly show exactly where some subjects had difficulty in adapting the source. One of the predictions from an analysis of Reed et al.'s text was that subjects would find problems mapping elements onto the relevant equation. Half of the subjects in the Table Condition managed to fill in the times and speeds in the table. However, fewer than 10% managed to map these values onto the correct equation (see figure 5.14). These results contrast with Reed and Ettinger's (1987) who found that their subjects were able to generate the equation from the values in the table when those values were given.

Although more subjects in experiment 3 managed to fill in the values in the table than in experiment 2, they were still unable to generate the equation. In fact, more subjects in experiment 2 managed to generate the correct equation in the Table/Distant Condition than in experiment 3. These results are perhaps surprising given the greater (presumed) expertise of the older subjects in Experiment 3. However, the explanations they had were less complete. This emphasizes yet again the importance of not making assumptions about the prior knowledge of the subjects.

Another notable aspect of the behaviour of the subjects was the degree to which they tried to imitate the example. Although it was predicted that some subjects would have difficulty in understanding where the $^3/_2$ came from in the example, and therefore that it would be hard to substitute the relevant value in the test problem, the lengths to which subjects went to adapt their solution to make it look like the example was quite marked. Those who saw the correspondence between the 2 in the test problem and the $^3/_2$ in the example still adapted it to make it *look* like $^3/_2$. This behaviour resembles the 'magic'

view of recursion described by Kahney (1982) whose subjects imitated an example involving recursion in SOLO with no real understanding of how it worked. The subjects in this experiment seemed to be employing the same strategy of imitating the practice problem in the belief that the closer the exercise problem resembled the example superficially the more likely they were to get a correct answer.

The reliance on mapping surface features is quite pronounced. Indeed, Novick and Holyoak (1991) fail to make the obvious point that providing subjects with 'number mappings' is in fact highlighting the relevant surface features of problems. Providing more structural aspects such as 'concept mappings' was less effective. The results from the present experiments, as well as those of Reed, Dempster and Ettinger, and Novick and Holyoak, demonstrate the need to show students as explicitly as possible how the surface features can be manipulated and mapped to the relevant equation; and how they can be adapted to fit variants of the problem type.

5. Understanding the text

On reading the text of the example problem subjects were asked to fill in boxes to indicate how well they thought they understood the problem explanation. In the Table Condition they were asked to rate their understanding of the following items (the corresponding bar in figure 5.16 is given in brackets after them where necessary):

the problem statement;
the distance-speed-time equation ('d = s x t' in figure 5.16);
the instruction to construct a table representing distance, speed and time for each car ('construct table');
the explanation of 't' and 't+3/2' ('t+3/2');
the speeds for the two cars ('mapping speeds');
the explanation of the need to represent distance by multiplying rate and time for each car ('represent distance in table');
the completed table;
the final equation and the explanation of how it is derived ('equation');
solving for 't' to derive the final solution ('solution').

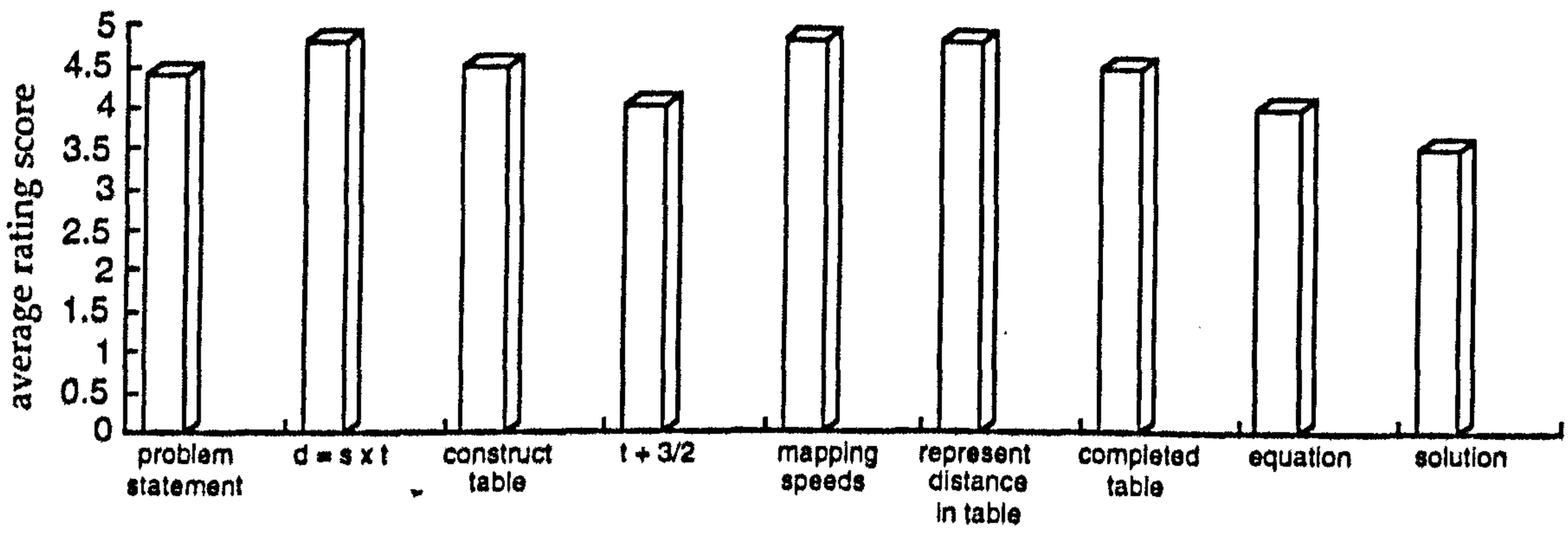


Figure 5.16. Average rating scores for 'understanding' in the Table Condition

(In the Table explanation there were three boxes alongside each line of the final solution. In the Mapping Condition only one rating was required. To make figures 5.16 and 5.17 comparable the three scores for the 'solution' in the Table condition have been averaged.)

In the Mapping Condition subjects were asked to rate the following aspects of the explanation:

the explanation that both cars travel the same distance so the *rate x time* for both will be equal since the distances are equal ('distances equal' in figure 5.17);

some values in the problem statement can be mapped to the equation ('mapped values');
the requirement to find any differences between the times and rates for the vehicles ('find differences');
the explanation that these derived differences can be mapped to the equation ('map derived values').

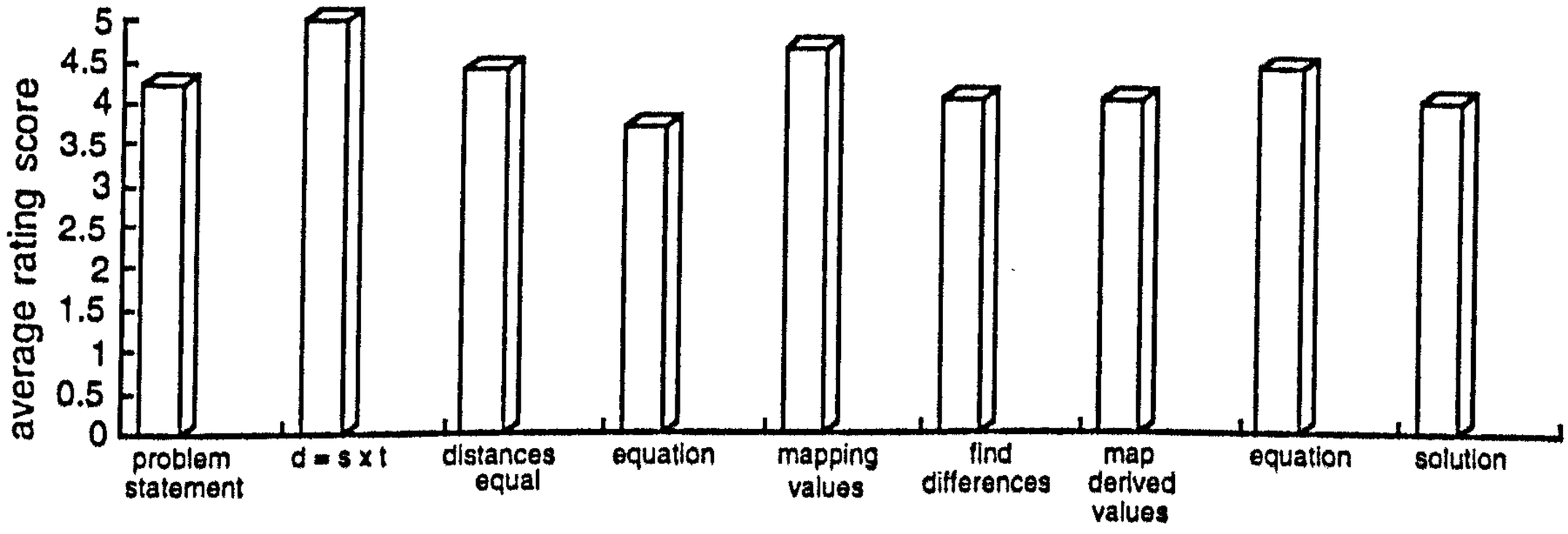


Figure 5.17. Average rating scores for 'understanding in the Mapping Condition.

Taking all the individual scores together, the average score in both conditions is virtually the same: 4.3 in the Table Condition and 4.2 in the Mapping Condition, corresponding to 'I think I understand this reasonably well'. The scores for each subject were calculated separately and compared with their performance on the transfer task (see Appendix 2 for the raw data). In the Table Condition the correlation between the subjects' assessment of their understanding and their performance was low ($r = 0.18$). In the Mapping Condition, on the other hand, the correlation was high ($r = 0.82$).

5.1. Problem understanding and performance

Both groups of subjects claimed to understand the explanations of their respective texts either 'reasonably well' or 'perfectly' in both the Table and Mapping conditions. The hypothesis that the subjects in the Table Condition would find it harder to understand the explanation they were given, since more inferences were involved, was not supported.

In the Table Condition, the subjects' own perception of how well they understood the explanation provided bore no relation to how well they did in the transfer task. There was no correlation between the actual scores of the subjects and their assessment of their understanding. In the Mapping Condition, on the other hand, there was a significant correlation. This latter result is unlikely to be because the subjects were better able to assess their own understanding than the subjects in the Table Condition. The difference must reside in the type of explanation provided for the example problems.

When subjects say they understand something, that understanding is confined to the specific example presented. They understand the problem but not necessarily how to adapt it *unless the problem explanation provides that kind of information.* If the example provides a *strategy* for dealing with problems of a particular type and subjects claim to understand it, then they are more likely to be able to solve a transfer problem. Obviously the two groups are not referring to the same things when they say they 'understand perfectly'. In the Table Condition subjects are presented with declarative knowledge relevant to the specific example. In the Mapping Condition subjects have more information about the procedure and how to adapt it; their assessment of how well they understand refers both to the declarative information about the specific example and about the general procedural information it gives, which is more pertinent to transfer than the former.

6. General Discussion

All of the predictions which were made from the analysis of the text of the distance-rate-time problems were confirmed in experiments 2 and 3. All areas where inferences were needed, where operators were not explicitly stated and where mappings were 'blocked' caused a significant number of errors. When those inferences were removed and the relation between the problem givens and their role in the solution was made explicit, significantly fewer errors were made. When information was provided about how to adapt the problem, subjects were better able to solve a 'distant' variant of the problem type. If subjects were using IPS, then any variation, however minor, was predicted to be a potential source of error. The resuts of experiments 2 and 3 showed that the particular kinds of error can be predicted from an analysis of the example problem and how it maps onto the target. IPS assumes that subjects attempt to map corresponding values from one problem to another. The next step is to infer what operations to apply to the mapped values in a step-by-step fashion by comparing the source to the target. Subjects often had to infer how one step was transformed into another in Reed et al.'s example and then had to apply that transformation in the target problem. When information was supplied about what transformations to apply to guide the adaptation of the problem, transfer was much more successful.

When more general information was provided, for example, about the need to find differences between the speeds and times of the vehicles, that information could be used to guide their adaptation of the example problem, leading to more successful transfer.

Other evidence for a heavy reliance on imitation as a problem solving strategy was shown by subjects who unnecessarily adapted the exercise problem to make it look more like the practice problem. The overt imitation and over-transfer exhibited by the subjects shows how greatly they relied on the surface features of problems. In experiment 2, even those subjects who successfully solved the problem had acquired only the vaguest representation of the solution procedure from the example problem, since most correct answers in the Table Condition also included the $^4/_2$ instead of 2. Such 'superstitious' behaviour suggests that the subjects did not have a clear understanding of the example problem and were merely copying the procedure in the hope, rather than the knowledge, that it would work. Subjects who have only a 'primary' understanding of the problem concentrate on its surface features. They do not necessarily understand it at a deeper or more abstract level. Their understanding is context specific.

In the field of pedagogy, a shift is needed from labelling subjects as 'good' or 'poor' performers on the basis of how they process and understand explanations in textbooks. This is not to say that individual differences do not exist. It is just more useful to shift the focus of attention to the structure of the expository text rather than looking at the 'failure' of particular subjects to understand. This shift of focus might help us advance our understanding of problem solving behaviour, and may help maximize the rate at which *all* students learn.

Chapter 6 CONCLUSIONS

The central argument of this thesis has been that the structure of a textbook and the examples it provides strongly influence the behaviour of novices as they use them to solve exercise problems. This chapter goes over the main arguments adduced to justify regarding imitation as the predominant process in much of novice problem solving from textbook examples. I review how the interpretation theory can reveal the relationship between the structure of source and target problems on one hand, and the likely behaviour of the solver on the other. The chapter concludes with a discussion of some of the implications of this research and suggestions for further study.

1. The arguments for the importance of imitation in problem solving

By virtue of their lack of domain-relevant knowledge, novices rely on surface features of examples to solve test problems. The process they use in so doing has been referred to as Imitative Problem Solving. By using imitation, novices can map the surface features of the problems and apply the transformations that took place in the earlier problem to the new one. Where the test problem is a close variant of the example, imitation will often yield a successful result. The more an example has to be adapted, the less likely it is that novices will be able to solve the test problem.

Several reasons were advanced as to why we should regard imitation as the prime method for problem solving by novices. First of all, novices have not yet developed the domain-specific schemas necessary to organize, interpret and integrate new information. Lacking

the necessary schemas for making sense of new material, novices tend to process expository material in a superficial manner.

Secondly, the way novices represent problems limits their ability to make inferences and adapt the source problem to solve a target. The representations novices have of both an exercise problem and any relevant source example are likely to be incomplete, and fragmentary. When novices come to do exercise problems later, they have to rely on imitating the examples. It is the only information they have to go on. For this reason a variant of a problem type that is even slightly different from the example will cause difficulties for novices.

Thirdly, I argued that the structure of problems, which may well be analogous, may not be known to, or understood by, the novice. This seems to fly in the face of the evidence that analogies are powerful learning devices (e.g. Gentner, 1983; Anderson & Thompson, 1989). However, the argument depends on the purpose of the analogy and its nature. For example, analogies in texts can often be very useful pedagogical devices. Expository analogies have been shown to enhance subjects' ability to generate inferences in a new domain (e.g. Donnelly & McDaniel, 1993; Gentner & Gentner, 1983). However, they work only when the subject already has the relevant prior knowledge of the base domain. The studies into the processes involved in APS carried out by Gick & Holyoak (1980; 1983) went one step further in the sense that both the base domain and the target were unknown to the subject prior to the studies. That is, the base domain had to be taught before subjects were presented with the target. This is also the case in scientific domains where new concepts and examples are first taught before exercise problems are presented. Nevertheless, problems such as the Fortress and Radiation problems have a *relatively* simple structure which is easy to recall. The relations involved are ones we already know; that is, we can understand the motivations of the protagonists, we know what it means to divide an army into groups, we can see that simultaneously converging on the fortress means that the whole of the original strong force is brought to bear, and so on. In scientific domains the nature of the relations between the elements of a problem may not be well known or understood.

When the base problem contains novel concepts and there is a complex network of relations between them, as is the case in most science texts, we cannot assume that novices can readily understand the explanation of example solutions. They may not be aware of the underlying structure of the example problem. If this is the case they cannot use structural analogy to solve a new problem of the same kind. Successfully solving a new problem which is a close variant of the example can be accomplished by imitating it. This does not require an understanding of the underlying structure.

A fourth argument concerned the nature of imitation as a 'useful' cognitive process. In solving problems, human beings will tend to take a path that involves the least cognitive effort. Very often the simplest way of solving a problem is to think of a similar one we have solved in the past and use that solution. Repeating a sequence of actions which was successful in achieving one's goals is a rational strategy, as long as the context remains more or less the same. Imitating a sequence of actions performed by others is likewise a useful strategy. So imitating a textbook example should normally result in a successful solution, if the target problem and source are seen as similar.

By relying on the perceptually similar features of problems novices make the reasonable assumption that the underlying structural or conceptual features are likely to be similar also. Solving a problem from an understanding of the structure of the problem is computationally demanding; it is simpler to imitate an earlier one. In terms of cognitive effort, imitation is thus an economical strategy.

A fifth argument adduced was that induction is conservative. Novices are sensitive to the specific contexts in which the examples are embedded. Their ability to generalize from those contexts is often limited because they may not know what inferences are justified or appropriate in a different context. This is the principle of 'conservative induction' (Medin & Ross, 1989).

2. Imitative Problem Solving and the interpretation theory

If novices imitate an example without fully understanding its structure, this has important consequences for how we investigate the processes they use. The interpretation theory introduced in Chapter 3 described a methodology that allows us to see both the structure of textbook examples and how they map onto test problems. The degree to which they map is predictive of the difficulty that novices have in imitating examples to solve a test problem. It also provides a metric of transfer. We can simply count the places where inferences have to be made, either because the explanation is not comprehensive, or because the two problems do not map directly. The more inferences that have to be made, the more difficult it is to adapt the problem to solve a target[1] . The interpretation theory,

---

[1] *It goes without saying that some inferences are easier to make than others. In any particular area the relative difficulty of generating inferences is an empirical question. The research presented here emphasizes that* wherever *there are inferences there is likely to be a source of difficulty.*

when used to describe the relations between a problem statement and its solution and between an example problem and a test problem, provides an explanation for many of the findings in the literature about the difficulties of transfer between problems. Even though problems may have the same underlying principle or equation, novices are often asked to solve exercise problems which are *different* from what they were taught to solve.

Textbooks in formal domains such as mathematics, science and computer programming have a particular stereotypical layout (Beck & McKeown, 1989; Kieras, 1985; Sweller & Cooper, 1985). The interpretation theory provides a way of examining such texts at a coarse grain to illuminate the overall structure of the text and to indicate where the text may deviate from the structure that expository texts are presumed to have. Students come to such textbooks with certain expectations about how the material in the book will be presented. They are likely to have a *schema* for how such textbooks are organized. Each chapter usually contains information about novel concepts, rules and relations which are then illustrated using example problems and solutions. These are then followed by a number of exercise problems. When reading training texts and attempting subsequent problems, students are likely to assume that all the information relevant to the solution of the test problem has been presented. That is, the students may not expect to have to make many inferences when studying the training material for the first time. One of the uses to which the interpretation theory can be put is to examine how successfully a text conforms to this schema, and, as a corollary, where violation of the schema is likely to lead to error and misapprehension.

3. Implications for textbook design and suggestions for future research

The research reported here has concentrated on the structure of texts in formal domains such as science, mathematics and computer programming. It is not within the scope of the thesis to deal with specific aspects of text structure such as the use of summaries, advance organizers, headings[2], or the like (see Newton, 1990). It has not dealt with how the interpretation theory can be extended to cover other domains that have a less well defined structure, such as texts in the social sciences or the arts. The kinds of examples found in other areas are often much less well defined. It remains to be seen how far the

---

[2]*This does not mean that it is impossible for these aspects of text structure to be coded using the interpretation theory.*

theory can be 'stretched' to deal with less formal but nonetheless 'knowledge-rich' domains.

The studies reported in chapter 5 have shown that subjects' understanding of new material is often superficial, even though they claim to understand it 'perfectly'. They have also shown how the verbal explanations of certain problem types can be improved by removing the need to make inferences, and by adding more information about how the example relates to a range of problems of the same type. This does not mean that the 'improved' explanations were in any sense 'optimal', merely that they were better than the original ones. Nevertheless, this research underlines the fact that the relations between a problem statement and its solution should be presented as comprehensively as possible. Similarly, the relation between the example and a range of problems of the same type should also be explained. That is, the underlying schema or principle and how it relates to a range of problems should be explained as clearly as possible. For the same reason, the use of hints to refer to earlier problem should be given, since novices have often great difficulty in making the necessary text-reinstatement inferences. We cannot assume that the novices have learned or understood everything in earlier sections.

Much has been written about people's understanding of narrative texts. Yet given the vast body of technical prose that exists, probably outnumbering narrative prose in its variety, it is strange that it should have been relatively neglected until recently. The range of such technical prose is wide, from manuals on how to operate a VCR to textbooks on quantum field theory. A technological society demands a highly trained workforce. Since much of this training comes from textbooks, it is important we get them right. This thesis has shown that by improving the quality of textbook explanations and examples, we can increase the likelihood that students will learn from them and apply their learning to new contexts. Much of the onus is on the writer.

PAGE NUMBERING AS ORIGINAL

REFERENCES

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory & Cognition*, 9, 422-433.

Adelson, B. (1984). When novices surpass experts: The difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 10, 483-495.

Anderson, J. R. (1983). *The Architecture of Cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R. (1989). A theory of the origins of human knowledge. *Artificial Intelligence*, 40, 313-351.

Anderson, J. R. (1990). *The Adaptive Character of Thought*. Hillsdale, NJ: Erlbaum.

Anderson, J. R., Boyle, C. F., Corbett, A. T., & Lewis, M. W. (1990). Cognitive modeling and intelligent tutoring. In W. J. Clancey & E. Soloway (Eds.), Artificial Intelligence and Learning Environments Cambridge, MA: The MIT Press.

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill aquisition and the LISP tutor. *Cognitive Science*, 13, 467-505.

Anderson, J. R., Farrell, R., & Sauers, R. (1984). Learning to program in LISP. *Cognitive Science*, 8, 87-129.

Anderson, J. R., Pirolli, P., & Farrell, R. (1988). Learning recursive programming. In M. Chi, M. Farr, & R. Glaser (Eds.), The nature of expertise (pp. 153-183). Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J. R., & Thompson, R. (1989). Use of an analogy in a production system architecture. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning* (pp. 267-297). London: Cambridge University Press.

Barclay, J. R., Bransford, J. D., Franks, J. J., McCarrell, N. S., & Nitsch, K. E. (1974). Comprehension and semantic flexibility. *Journal of Verbal Learning and Verbal Behaviour*, 13, 471-481.

Barsalou, L. W. (1989). Intraconcept similarity and its implications for interconcept similarity. In S. Vosniadou & A. Ortony (Eds.), Similarity and Analogical Reasoning London: Cambridge University Press.

Bassok, M. (1990). Transfer of domain-specific problem solving procedures. *Journal of Experimental Psychology Learning, Memory, and Cognition*, 16(3), 522-533.

Bassok, M., & Holyoak, K. J. (1989). Interdomain transfer between isomorphic topics in algebra and physics. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 15, 153-166.

Baudet, S., & Denhière, G. (1991). Mental models and acquisition of knowledge from text: representation and acquisition of functional systems. In G. Denhière & J.-P. Rossi (Eds.), *Text and text processing* (pp. 155-187). Amsterdam:

Beck, I. L., & McKeown, M. G. (1989). Expository text for young readers: The issue of coherence. In L. B. Resnick (Eds.), *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser* (pp. 47-66). Hillsdale, NJ: Erlbaum.

Berwick, R. C. (1986). Learning from positive only examples: The subset principle and three case studies. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), Machine learning: An artificial intelligence approach Los Altos, CA: Morgan Kaufman.

Beveridge, M., & Parkins, E. (1987). Visual representation in analogical problem solving. *Memory and Cognition*, 15(3), 230-237.

Boden, M. A. (1972). Purposive Explanation in Psychology. Cambridge, MA: Harvard University Press.

Bonar, J., & Soloway, E. (1985). Preprogramming knowledge: A major source of misconceptions in novice programmers. *Human-Computer Interaction*, 1(2), 133-161.

Britton, B. K., Van Dusen, L., Glynn, S. M., & Hemphill, D. (1990). The impact of inferences on instructional text. In A. C. Graesser & G. H. Bower (Eds.), *The Psychology of Learning and Motivation: Inferences and Text Comprehension* (pp. 53-70). London: Academic Press.

Britton, B. K., Van Dusen, L., Gulgoz, S., & Glynn, S. M. (1989). Instructional texts rewritten by five expert teams: Revisions and retention improvements. *Journal of Educational Psychology*, (2), 226-239.

Brown, A. L. (1989). Analogical learning and transfer: What develops? In S. Vosniadou & A. Ortony (Eds.), Similarity and analogical reasoning Cambridge: Cambridge University Press.

Brown, A. L., & Kane, M. J. (1988). Preschool children can learn to transfer: Learning to learn and learning from example. Cognitive Psychology, 20, 493-523.

Brown, D. E., & Clement, J. (1989). Overcoming misconceptions via analogical reasoning: Abstract transfer versus explanatory model construction. *Instructional Science*, 18(4), 237-261.

Brown, J. S., & Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 2, 155-192.

Brown, J. S., & VanLehn, K. (1980). Repair Theory: A generative theory of bugs in procedural skills. *Cognitive Science*, 4, 379-426.

Byrnes, J. P. (1992). The conceptual basis of procedural learning. *Cognitive Development*, 7, 235-257.

Carbonell, J. G. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In R. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), Machine Learning: An Artificial Intelligence Approach Palo Alto, CA: Tioga Press.

Carroll, J. M., Smith-Kerker, P. L., Ford, J. R., & Mazur-Rimetz, S. A. (1987-1988). The Minimal Manual. *Human-Computer Interaction*, 3, 123-153.

Catrambone, R. (1990). Specific versus general procedures in instructions. *Human Computer Interaction*, 5, 49-93.

Catrambone, R., & Holyoak, K. J. (1989). Overcoming contextual limitations on problem-solving transfer. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 15(6), 1147-1156.

Chase, W. D., & Simon, H. A. (1973a). *The mind's eye in chess*. New York, NY: Academic Press.

Chase, W. D., & Simon, H. A. (1973). The mind's eye in chess. In W. G. Chase (Eds.), Visual information processing (pp. 215-282). New York, NY: Academic Press.

Chen, Z., & Daehler, M. W. (1989). Positive and negative transfer in analogical problem solving by 6 year old children. *Cognitive Development*, 4(4), 327-344.

Cheng, P. W., & Holyoak, K. J. (1985). Pragmatic reasoning schemas. *Cognitive Psychology*, 17(4), 391-416.

Chi, M. T. H., & Bassok, M. (1989). Learning from examples via self-explanations. In L. B. Resnick (Eds.), *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser* (pp. 251-282). Hillsdale, NJ: Erlbaum.

Chi, M. T., Bassok, M., Lewis, M. W., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13, 145-182.

Chi, M. T. H., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science*, 5, 121-152.

Chi, M. T. H., Glaser, R., & Rees, E. (1982). Expertise in problem solving. In R. J. Sternberg (Eds.), *Advances in the psychology of human intelligence* (pp. 7-75). Hillsdale, NJ: Lawrence Erlbaum Associates.

Clement, J. (1988). Observed methods for generating analogies in scientific problem solving. *Cognitive Science*, **12**, 563-586.

Clement, J. J. (1989). Generation of spontaneous analogies by students solving science problems. In D. M. Topping, D. C. Crowell, & V. N. Kobayashi (Eds.), *Thinking across cultures: The Third International Conference on Thinking* (pp. 303-308). Hillsdale, NJ: Erlbaum.

Collins, A., & Gentner, D. (1987). How people construct mental models. In D. Holland & N. Quinn (Eds.), *Cultural models in language and thought* (pp. 243-265). Cambridge: Cambridge University Press.

Collins, A., & Gentner, D. (1990). Multiple models of evaporation processes. In D. S. Weld De-Kleer, J. (Eds.), *Readings in qualitative reasoning about physical systems* (pp. 508 - 512). San Mateo, California: Morgan Kaufmann.

Conway, M., & Kahney, H. (1987). Transfer of learning in inference problems. In J. Hallam & C. Mellish (Eds.), *Advances in Artificial Intelligence* Chichester: John Wiley.

Cooper, G., & Sweller, J. (1987). Effects of schema acquisition and rule automation on mathematical problem solving transfer. *Journal of Educational Psychology*, **79**(4), 347-362.

Dejong, G. (1989). The role of explanation in analogy; or, the curse of an alluring name. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning* London: Cambridge University Press.

Dellarosa-Cummins, D. (1992). The role of analogical reasoning in the induction of problem categories. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **18**(5), 1103-1124.

Détienne, F. (1991). Reasoning from a schema and from an analog in software code reuse. In *Empirical studies of programmers: Fourth workshop*, . New Brunswick, NJ:

Donnelly, C. M., & McDaniel, M. A. (1993). Use of analogy in learning scientific concepts. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **19**(4), 975-987.

DuBoulay, B., O'Shea, T., & Monk, J. (1989). The black box inside the glass box: Presenting computing concepts to novices. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* Hillsdale, NJ: Erlbaum.

Duncker, K. (1945). On problem solving. *Psychological Monographs*, 58, (Whole no. 270).

Egan, D. E., & Greeno, J. G. (1974). Theory of rule induction: Knowledge acquired in concept learning, serial pattern learning, and problem solving. In L. Gregg (Eds.), *Knowledge and Cognition* (pp. 43-103). Hillsdale, NJ: Lawrence Erlbaum.

Ehrlich, K., & Soloway, E. (1984). An empirical investigation of tacit plan knowledge in programming. In J. C. Thomas & M. L. Schneider (Eds.), *Human Factors in Computer Systems* Norwood, NJ: Ablex.

Eisenstadt, M. (1978/1983). *Units 3-4, Artificial Intelligence Project*. D303 Cognitive Psychology, Milton Keynes: The Open University.

Eisenstadt, M. (1983). A user friendly software environment for the novice programmer. *Communications of the Association for Computing Machinery*, 27, (12), 1056-1064.

Eylon, B.-S., & Reif, F. (1984). Effects of knowledge organization on task performance. *Cognition and Instruction*, Win Vol 1(1), 5-44.

Falkenhainer, B., Forbus, K. D., & Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41, 1-63.

Ferguson-Hessler, M. G. M., & Jong, T. d. (1990). Studying physics texts: Differences in study processes between good and poor performers. *Cognition and Instruction*, 7(1), 41-54.

Fried, L. S., & Holyoak, K. J. (1984). Induction of category distributions: A framework for classification learning. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 10, 234-257.

Gentner, D. (1989). The mechanisms of analogical reasoning. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning* London: Cambridge University Press.

Gentner, D., & Gentner, D. R. (1983). Flowing waters or teeming crowds: mental models of electricity. In D. Gentner & A. L. Stevens (Eds.), *Mental models* Hillsdale, N.J.: Lawrence Erlbaum Associates.

Gentner, D., & Landers, R. (1985). Analogical reminding: a good match is hard to find. In *Proceedings of international conference on systems, man and cybernetics,* (pp. 607-613). Tucson, AZ: IEEE.

Gentner, D., & Rattermann, M. J. (1991). Language and the Career of similarity. In S. E. Gelman & J. P. Byrnes (Eds.), *Perspectives on Language and Thought: Interrelations in Development* (pp. 225-277). London: Cambridge University Press.

Gentner, D., & Schumacher, R. M. (1987). Use of structure mapping theory for complex systems. In *Proceedings of the 1986 IEEE international conference on systems, man and cybernetics,* **Vol 1** (pp. 252 - 258). Piscataway, NJ: IEEE.

Gentner, D., & Stevens, A. L. (1983). *Mental Models.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Gentner, D., & Toupin, C. (1986). Systematicity and surface similarity in the development of analogy. *Cognitive Science,* **10**, 277-300.

Gick, M. L. (1985). The effect of a diagram retrieval cue on spontaneous analogical transfer. *Canadian Journal of Psychology,* **39**(3), 460-466.

Gick, M. L., & Holyoak, K. J. (1980). Analogical problem solving. *Cognitive Psychology,* **12**, 306-356.

Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. *Cognitive Psychology,* **15**, 1-38.

Glaser, R. (1984). The role of knowledge. *American Psychologist,* **39**(2), 93-104.

Glaser, R., & Bassok, M. (1989). Learning theory and the study of instruction. In M. R. Rosenzweig & L. W. Poerter (Eds.), *Annual Review of Psychology* (pp. 631-666). Palo Alto, CA: Annual Reviews, Inc.

Gold, K. (1992). If all else fails, read the instructions. *New Scientist,* 13 June, 38-41.

Gordon, J. E. (1976). *The New Science of Strong Materials: or Why You Don't Fall Through the Floor.* Harmondsworth, England: Penguin.

Goswami, U. (1992). *Analogical Reasoning in Children.* Hove, UK: Erlbaum.

Green, A. J. K. (1989) *Statistical computing: Individual differences in the acquisition of a cognitive skill.* Unpublished doctoral thesis. University of Aberdeen.

Green, A. J. K., & Gilhooly, K. J. (1990a). Individual differences and effective learning procedures: The case of statistical computing. *International Journal of Man-Machine Studies*, 33, 97-119.

Green, A. J. K., & Gilhooly, K. J. (1990b). Statistical computing: Individual differences in learning at macroscopic and microscopic levels. In K. J. Gilhooly, M. T. G. Keane, R. H. Logie, & G. Erdos (Eds.), *Lines of Thinking* Chichester: Wiley.

Grudin, J. (1980). Processes in verbal analogy solution. *Journal of Experimental Psychology: Human Perception & Performance*, 6, 67-74.

Hall, R., Kibler, D., Wenger, E., & Truxaw, C. (1989). Exploring the episodic structure of algebra story problem solving. *Cognition and Instruction*, 6(3), 223-283.

Hasemer, T., & Domingue, J. (1989). *Common Lisp Programming for Artificial Intelligence.* Wokingham: Addison-Wesley.

Heller, J. I. (1979) *Cognitive processes in verbal analogy solutions.* Unpublished doctoral thesis, University of Pittsburgh.

Hiebert, J. (Ed.). (1986). *Conceptual and procedural knowledge: The case of mathematics.* Hillsdale, NJ: Erlbaum.

Hiebert, J., & Lefevre, P. (1986). Conceptual and procedural knowledge in mathematics: An introductory analysis. In J. Hiebert (Eds.), *Conceptual and Procedural Knowledge: The Case of Mathematics* (pp. 1-27). Hillsdale, NJ: Erlbaum.

Holderness, J. (1987). *GCSE Maths: Higher Level.* Ormskirk, Lancs.: Causeway Books.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1986). *Induction: Processes of Inference, Learning and Discovery.* Cambridge, MA: MIT Press.

Holyoak, K. J. (1984). Analogical thinking and human intelligence. In R. J. Sternberg (Eds.), *Advances in the Psychology of Human Intelligence* Hillsdale, NJ: Erlbaum.

Holyoak, K. J. (1985). The pragmatics of analogical transfer. In G. H. Bower (Eds.), *The Psychology of Learning and Motivation* New York: Academic Press.

Holyoak, K. J., Junn, E. N., & Billman, D. O. (1984). Development of analogical problem-solving skill. *Child Development*, 55(6), 2042-2055.

Holyoak, K. J., & Koh, K. (1987). Surface and structural similarity in analogical transfer. *Memory and Cognition*, 15(4), 332 340.

Holyoak, K. J., & Thagard, P. R. (1989a). A computational model of analogical problem solving. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning* London: Cambridge University Press.

Holyoak, K. J., & Thagard, P. (1989b). Analogical mapping by constraint satisfaction. *Cognitive Science*, 13(3), 295-355.

Howat, R. D., Mullan, E. C. K., Nisbet, K., & Brown, D. (1987). *Mathematics in Action.* Glasgow: Blackie & Son.

Issing, L. J., Hannemann, J., & Haack, J. (1989). Visualization by pictorial analogies in understanding expository text. In H. Mandl & J. R. Levin (Eds.), *Knowledge Acquisition from Text and Picture* (pp. 195-214). Amsterdam: Elsevier.

Johnson-Laird, P. N. (1983). *Mental models: towards a cognitive science of language inference and conciousness.* Cambridge: Cambridge University Press.

Johnson-Laird, P. N. (1989). Analogy and the exercise of creativity. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning* London: Cambridge University Press.

Kahney, H. (1982). *An in-depth study of the cognitive behaviour of novice programmers* (HCRL Technical Report No. 5). The Open University.

Kahney, H. (1993). *Problem Solving: Current Issues* (2nd ed.). Milton Keynes: Open University Press.

Kessler, C. M., & Anderson, J. R. (1989). Learning flow of control: recursive and iterative procedures. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* Hillsdale, NJ: Erlbaum.

Kieras, D. E. (1985). Thematic processes in the comprehension of technical prose. In B. K. Britton & J. B. Black (Eds.), *Understanding Expository Text: A Theoretical and Practical Handbook for Analyzing Explanatory Text* (pp. 89-107). Hillsdale, NJ: Erlbaum.

Kinstch, W. (1986). Learning from text. *Cognition and Instruction*, 3(2), 87-108.

Kintsch, W., & Greeno, J. G. (1985). Understanding and solving word arithmetic problems. *Psychological Review*, 92(1), 109-129.

Kintsch, W., & Van Dijk, T. A. (1978). Toward a model of text comprehension and production. *Psychological Review*, 85, 363-394.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33, 1-64.

Larkin, J. H. (1978). *Problem solving in physics: Structure, process and learning*. The Netherlands: Sijthoff and Noordhoff.

Larkin, J. H. (1989). What kind of knowledge transfers? In L. B. Resnick (Eds.), *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser* (pp. 283-306). Hillsdale, NJ: Erlbaum.

Larkin, J. H., McDermott, J., Simon, D. P., & Simon, H. A. (1980). Expert and novice performance in solving physics problems. *Science*, 28, 1335-1342.

Larkin, J. H., & Simon, H. A. (1987). Why a diagram is sometimes worth ten-thousand words. *Cognitive Science*, 11, 65-99.

LeFevre, J., & Dixon, P. (1986). Do written instructions need examples? *Cognition and Instruction*, 3, 1-30.

LeFevre, J. A. (1987). Processing instructional texts and examples. *Canadian Journal of Psychology*, 41(3), 351-364.

Levin, J. R. (1988). A transfer-appropriate processing perspective of pictures in prose. In H. Mandl & J. R. Levin (Eds.), *Knowledge acquisition from text and pictures* Amsterdam, Netherlands: North Holland.

Mathews, R. C., Buss, R. R., Stanley, W. B., Blanchard-Fields, F., & et-al (1989). Role of implicit and explicit processes in learning from examples: a synergistic effect. *Journal of Experimental Psychology Learning, Memory, and Cognition*, 15(6), 1083-1100.

Mayer, R. E. (1989). The psychology of how novices learn computer programming. In E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer* Hillsdale, NJ: Erlbaum.

Medin, D. L., & Ortony, A. (1989). Psychological essentialism. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning* London: Cambridge University Press.

Medin, D. L., & Ross, B. H. (1989). *The specific character of abstract thought: categorization, problem solving and induction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Morris, C. D., Bransford, J. D., & Franks, J. J. (1977). Levels of processing versus transfer appropriate behaviour. *Journal of Verbal Learning and Verbal Behavior*, 16, 519-533.

Nathan, M. J., Kintsch, W., & Young, E. (1992). A theory of algebra-word-problem comprehension and its implications for the design of learning environments. Cognition and Instruction, 9(4), 329-389.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.

Newell, A., & Simon, H. A. (1972). *Human Problem Solving*. New Jersey: Prentice Hall.

Newton, D. P. (1990). *Teaching with Text : Choosing, Preparing and Using Textual Materials for Instruction*. London: Kogan Page.

Norman, D. A., & Bobrow, D. G. (1979). Descriptions: An intermediate stage in memory retrieval. *Cognitive Psychology*, 11, 107-123.

Novick, L. R. (1988). Analogical transfer, problem similarity, and expertise. *Journal of Experimental Psychology Learning, Memory, and Cognition*, 14(3), 510-520.

Novick, L. R. (1990). Representational transfer in problem solving. *Psychological Science*, 1(2), 128-132.

Novick, L. R., & Holyoak, K. J. (1991). Mathematical problem solving by analogy. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 17(3), 398-415.

Ohlsson, S., & Rees, E. (1991). The function of conceptual understanding in the learning of arithmetic procedures. *Cognition and Instruction*, 8(2), 103-179.

Owen, E., & Sweller, J. (1985). What do students learn while solving mathematics problems? *Journal of Educational Psychology*, 77(3), 272-284.

Palmer, S. E. (1989). Levels of description in information-processing theories of analogy. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning* London: Cambridge University Press.

Pazzani, M. (1991). A computational theory of learning causal relationships. *Cognitve Science*, 15(3), 401-424.

Perry, M. (1991). Learning and transfer: Instructional conditions and conceptual change. *Cognitive Development*, 6, 449-468.

Phye, G. D. (1991). Advice and feedback during cognitive training: Effects at acquisition and delayed transfer. *Contemporary Educational Psychology*, 16(1), 87-94.

Pirolli, P. (1991). Effects of examples and their explanations in a lesson on recursion: A production system analysis. *Cognition and Instruction*, 8(3), 207-259.

Pirolli, P. L., & Anderson, J. R. (1985). The role of learning from examples in the acquisition of recursive programming skills. Special Issue: Skill. *Canadian Journal of Psychology;*, 39(2), 240-272.

Posner, M. I., & Keele, S. W. (1968). On the genesis of abstract ideas. *Journal of Experimental Psychology*, 77, 353-363.

Posner, M. I., & Keele, S. W. (1970). Retention of abstract ideas. *Journal of Experimental Psychology*, 83, 304-308.

Rayner, D. (1990). *Complete Mathematics for G.C.S.E. and Standard Grade*. Oxford: Oxford University Press.

Reber, A. S. (1989). Implicit learning and tacit knowledge. *Journal of Experimental Psychology: General*, 118(3), 219-235.

Reed, S. K. (1989). Constraints on the abstraction of solutions. *Journal of Educational Psychology*, 81(4), 532-540.

Reed, S. K., Ackinclose, C. C., & Voss, A. A. (1990). Selecting analogous problems: Similarity versus inclusiveness. *Memory and Cognition*, 18(1), 83-98.

Reed, S. K., & Bolstad, C. A. (1991). Use of examples and procedures in problem solving. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 17, 753-766.

Reed, S. K., Dempster, A., & Ettinger, M. (1985). Usefulness of analogous solutions for solving algebra word problems. *Journal of Experimental Psychology Learning, Memory, and Cognition*, 11(1), 106-125.

Reed, S. K., Ernst, G. W., & Banerji, R. (1974). The role of analogy in transfer between similar problem states. *Cognitive Psychology*, 6, 436-450.

Reed, S. K., & Ettinger, M. (1987). Usefulness of tables for solving word problems. *Cognition and Instruction*, 4(1), 43-58.

Resnick, L. B. (1989). Introduction to Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser. In L. B. Resnick (Eds.), *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser* (pp. 1-24). Hillsdale, NJ: Erlbaum.

Richardson, K., & McCarthy, T. (1990). The abstraction of covariation in conceptual representation. *British Journal of Psychology*, 81, 415-438.

Rips, L. J. (1989). Similarity, typicality and categorization. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning* London: Cambridge University Press.

Robertson, S. I., & Kahney, H. (1993). *How do examples help solvers solve problems? An Interpretation Theory for text analysis* No. 96). HCRL, The Open University.

Robertson, W. C. (1990). Detection of cognitive structure with protocol data: Predicting performance on physics transfer problems. *Cognitive Science*, 14(2), 253 280.

Ross, B. H. (1984). Remindings and their effects in learning a cognitive skill. *Cognitive Psychology*, **16**, 371-416.

Ross, B. H. (1987). This is like that: The use of earlier problems and the separation of similarity effects. *Journal of Experimental Psychology Learning, Memory, and Cognition*, **13**(4), 629-639.

Ross, B. H. (1989a). Distinguishing types of superficial similarities. Different effects on the access and use of earlier problems. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, **14**, 510-520.

Ross, B. H. (1989b). Remindings in Learning and Instruction. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning* (pp. 438-469). London: Cambridge University Press.

Ross, B. H., & Kennedy, P. T. (1990). Generalizing from the use of earlier examples in problem solving. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, **16**, 42-55.

Ross, B. H., & Sofka, M. D. (1986). *Remindings: Noticing, remembering and using specific knowledge of earlier problems*. Unpublished manuscript, University of Illinois, Department of Psychology, Urbana-Champaign

Sanderson, P. M. (1989). Verbalizable knowledge and skilled task performance: Association, dissociation, and mental models. *Journal of Experimental Psychology: Learning, Memory and Cognition*, **15**(4), 729-747.

Schank, R. C. (1982). *Dynamic Memory: A theory of reminding and learning in computers and people*. Cambridge, MA: Cambridge University Press.

Schumacher, R. M., & Gentner, D. (1988). Transfer of training as analogical mapping. *IEEE transactions on systems, man, and cybernetics*, **18**(4), 592 - 600.

Silver, E. A. (1986). Using conceptual and procedural knowledge: A focus on relationships. In J. Hlebert (Eds.), *Conceptual and Procedural Knowledge: The Case of Mathematics* (pp. 181-198). Hillsdale, NJ: Erlbaum.

Simon, P. R. J. (1984). Instructing with analogies. *Journal of Educational Psychology*, **76**(3), 513-527.

Skemp, R. (1978). Relational and instrumental understanding. *Arithmetic Teacher*, 26, 9-15 (quoted in Silver, 1986).

Smith, E. E., & Medin, D. L. (1981). *Categories and concepts*. Cambridge, MA: Harvard University Press.

Spearman, C. (1923). *The nature of "intelligence" and the principles of cognition*. London: Macmillan.

Spencer, R. M., & Weisberg, R. W. (1986). Context dependent effects on analogical transfer. *Memory and Cognition*, 14(5), 442-449.

Stein, B. S., Way, K. R., Benningfield, S. E., & Hedgecough, C. A. (1986). Constraints on spontaneous transfer in problem solving tasks. *Memory and Cognition*, 14(5), 432-441.

Sternberg, R. J. (1977). *Intelligence, information processing and analogical reasoning: The component analysis of human abilities*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Sweller, J. (1980). Transfer effects in a problem solving context. *Quarterly Journal of Experimental Psychology*, 32, 233-239.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12, 257-285.

Sweller, J., & Cooper, G. A. (1985). The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction*, 2, 59-89.

Sweller, J., Mawer, R. F., & Ward, M. R. (1983). Development of expertise in mathematical problem solving. *Journal of Experimental Psychology: General*, 112, 639-661.

Tardieu, H., Ehrlich, M.-F., & Gyselinck, V. (1992). Reader's knowledge and the control of inferences in reading. *Language and Cognitive Processes*, 7, 335-351.

Thorndike, E. L., & Woodworth, R. S. (1901). The influence of improvement in one mental function upon the efficiency of other functions. *Psychological Review*, 8, 247-261.

Tulving, E., & Thomson, D. M. (1973). Encoding specificity and retrieval processes in episodic memory. *Psychological Review*, 80, 352-373.

Van Dijk, T. A., & Kintsch, W. (1983). *Strategies of discourse comprehension*. New York, NY: Academic Press.

VanLehn, K. (1986). Arithmetic procedures are induced from examples. In J. Hiebert (Eds.), *Conceptual and procedural knowledge: The case of mathematics* Hillsdale, NJ: Erlbaum.

VanLehn, K. (1989). Problem solving and cognitive skill acquisition. In M. I. Posner (Eds.), *Foundations of Cognitive Science* Cambridge, MA: MIT Press.

VanLehn, K. (1990). *Mind Bugs: The Origins of Procedural Misconceptions*. Cambridge, MA: MIT Press.

Vosniadou, S. (1989). Analogical reasoning as a mechanism in knowledge acquisition: A developmental perspective. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning* London: Cambridge University Press.

Ward, M., & Sweller, J. (1990). Structuring effective worked examples. *Cognition and Instruction*, 7(1), 1-39.

Weisberg, R., DiCamillo, M., & Phillips, D. (1978). Transferring old associations to new situations: A nonautomatic process. *Journal of Verbal Learning and Verbal Behaviour*, 17, 219-228.

Wertheimer, M. (1959). *Productive thinking*. New York, NY: Harper and Row.

Whittlesea, B. W. A. (1987). Preservation of specific experiences in the representation of general knowledge. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 13, 3-17.

Winston, P. H. (1980). Learning structural descriptions from examples. In P. H. Winston (Eds.), *The psychology of computer vision* New York, NY: McGraw-Hill.

Winston, P. H., & Horn, B. K. P. (1981). *LISP*. Reading, MA: Addison-Wesley.

Young, R. M., & O'Shea, T. (1981). Errors in children's subtraction. *Cognitive Science*, 5, 153-177.

APPENDICES

Appendix 1 Experiment 1

A.1.1. Instructions for experiment 1

Sheet 1. Read this sheet and in the boxes under column 1 write a number from 1 to 5 to show how well you think you have understood the statements or the PROLOG clauses according to the following code:

5: Understand perfectly gist
4: Understand reasonably well
3: I think I have got the gist
2: Don't quite understand
1: Don't understand at all

A hierarchy of animals

FACTS	col. 1	col. 2
This part states which categories of things are animals, examples of category members and more specific examples:	☐	☐
isa(mammal animal) isa(bird animal). isa(fish animal). isa(trout fish). isa(Trigger horse). isa(Butch dog). isa(Jaws fish).	☐	☐
isa(dog mammal). isa(bat mammal). isa(shark fish). isa(albatross bird).		
This part gives the features of categories mentioned above and some features of some specific examples:	☐	☐
has(mammal milk). has(mammal fur). has(bird feathers). has(bird wings). has(fish gills). has(fish fins).	☐	☐
has(dog tail). has(dog teeth). has(shark dorsal_fin). has(horse hooves). has(horse mane). has(albatross big_wings).		
RULE		
This rule states that if something is an example of a category (for instance Trigger is an example of a horse) then it has the same features as other members of that category (that is, if Trigger is a horse then Trigger has hooves and a mane and has fur, etc)	☐	☐
A has(_Something _Some-feature) if isa(_Something _Category-member) & has(_Category-member _Some-feature).	☐	☐

Sheet 1. Read this sheet and in the boxes under column 1 write a number from 1 to 5 to show how well you think you have understood the statements or the PROLOG clauses according to the following code:

5: Understand perfectly gist
4: Understand reasonably well
3: I think I have got the gist
2: Don't quite understand
1: Don't understand at all

A Train Timetable

FACTS	**col. 1**	**col. 2**
This part states which trains leave from which stations:	☐	☐
departs(Train1 GLASGOW). departs(Train2 DUNDEE). departs(Train3 ABERDEEN). departs(Train4 BIRMINGHAM). departs(Train5 LONDON).	☐	☐
This part states which trains arrive at which stations:	☐	☐
arrives(Train1 BIRMINGHAM). arrives(Train2 GLASGOW). arrives(Train3 DUNDEE). arrives(Train4 LONDON). arrives(Train5 DOVER).	☐	☐
RULES *This rule states that if the same train leaves Station1 and arrives at Station2 then Station1 and Station2 are directly linked:*	☐	☐
B direct-link(_Station1 _Station2) if departs(_Train _Station1) & arrives(_Train _Station2).	☐	☐
This rule states that you can get from one station to another if these stations are linked either directly or there are other linked stations on the way:	☐	☐
C connected(_Station1 _Station2) if direct-link(_Station1 _Station2). connected(_Station1 _Station3) if direct-link(_Station1 _Station2) & connected(_Station2 _Station3).	☐	☐

Sheet 3. Read this sheet and in the boxes write a number from 1 to 5 to show how well you think you have understood the statements or the PROLOG clauses according to the following code:

5: Understand perfectly gist | 4: Understand reasonably well | 2: Don't quite understand | 3: I think I have got the | 1: Don't understand at all

---

Food Preferences

FACTS **col. 1**

Here is a database of food 'facts': ☐

```
isa(meat food).          isa(beef meat).
isa(poultry food).       isa(chicken poultry).
isa(vegetables food).    isa(root vegetable).
isa(pulses food).        isa(leaf vegetable).
isa(grain food).         isa(trout fish).
isa(fish food).  isa(salmon fish).
isa(carrot root).        isa(lentils pulses).
isa(potato root).        isa(apple fruit).
isa(lettuce leaf).       isa(rice grain).
```

isa(cabbage leaf). ☐

Here is what vegetarians and carnivores eat: ☐

```
eat(vegetarians vegetables).
eat(vegetarians pulses).
eat(vegetarians fruit).
eat(vegetarians grain).
eat(carnivores meat).
eat(carnivores fish).
```

eat(carnivores poultry). ☐

1) Now suppose that you were given the problem to write a rule or rules to represent what it is that omnivores eat. Think about this for a few minutes and then look at the other two sheets with the **train timetables** and the **hierarchy of animals** and, in the second column of boxes, tick anything that you think would help you solve the problem.

2) Choose between rules A, B and C on the other two sheets which one you think would be the most useful to you in solving the problem and write your answer in this box: ☐

3) Which of these do you think would be the most likely first line of your rule (tick the appropriate box):

☐ eat(_X food) if ...; ☐ eat(_Omnivores _X) if ...;

☐ eat(omnivores _X) if ...; ☐ eat(omnivores food) if ...

4) Are you acquainted with Prolog or any other artificial intelligence language outside the D309 course? (delete which does not apply) **Yes / No**

5) (Optional) Try writing the rule. If you do so, could you please tick everything that you refer to on these sheets while solving the problem adding a tick every time you do so.

A.1.2 Tabulation of results (experiment 1)

E.D1	Explanation of Database 1	D1	Database 1
E.R1	Explanation of Rule 1	R1	Rule 1
R2a	The first part of Rule 2	R2b	The second part of Rule 2
Mod	The rule chosen as a model for the omnivore problem		
Ist 1	The first line chosen for the omnivore problem		
AI	Acquaintance with AI language outside the D309 course		

	SHEET 1						SHEET 2									SHEET 3				TEST		
subj	E.D 1	D1	E.D 2	D2	E.R 1	R1	E.D 1	D1	E.D 2	D2	E.R 1	E1	E.R 2	R2a	R2b	E.D 1	D1	E.D 2	D2	Mod	Ist 1	AI
S1	4	5		5	4	4	5	5	5	5	5	5	5	5	3	5	5	5	5	A	3	?
S2	4	5	5	5	5	4	5	5	5	5	5	5	5	5	5	5	5	5	5	A	3	
S3	4	5	4	5	4	3	5	5	5	5		4	5	5	5	5	5	4	5	C	3	
S4	4	5	4	5	5	5	5	5	5	5	4	5	5	5	5	5	5	5	5	A	3	
S5	5	5	5	5	5	2	5	5	5	5	5	5	5	5	5	5	5	5	5	A	3	
S6	3	5	4	5	4	4	5	5	5	5	5	5	4	5	4	5	5	2	5	A	3	
S7	4	4	4	4	4	3	5	5	5	5	4	3	3	4	3	5	4	5	5	B	3	
S8	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	n/a	3	
S9	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	A	2	
S10	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	A	3	
S11	5	4	5	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	C	1	Y
S12	5	5	5	5	5	4	5		5		4	4	4	4		5	5	5	5	n/a	2	
S13	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5		5	5	B	3	
S14	5	5	5	5	4	4	5	5	5	5	5	5	5	5	5	5	5	5	5	A	1	
S15	5	4	5	5	5	4	5	5	5	5	5	4	4	4	3	5	5	5	5	A	3	
S16	4	4	4	4	3	2	4	4	4	4	4	4	2	2	2	4	4	4	4		3	
TOT	72	76	70	77	73	64	79	74	79	74	71	74	72	74	65	79	73	75	79			
	4.50	4.75	4.67	4.81	4.56	4.00	4.94	4.93	4.94	4.93	4.73	4.63	4.50	4.63	4.33	4.94	4.87	4.69	4.94			

Table A.1.1. Responses to PROLOG questionnaire.

Appendix 2. Experiments 2 and 3

A.2.1. Results tables for experiment 2

A.2.1.1. Codings for results in experiments 2 and 3

Table -> equation. In the Table Conditions the subjects' solutions are coded depending on whether the table has been completely and correctly filled, partially filled (usually the *time* and *speed* columns only), and whether the values in the boxes have been mapped on to the correct equation. They are coded according to the following scheme:

f	-	means that the table was correctly filled;
pf	-	means that the table was partially filled (i.e. the 'distance' column was incorrectly filled or was not completed);
wf	-	means that the table was filled incorrectly or boxes were not filled;
m	-	means that the elements were correctly mapped to the equation;
rm	-	means that the elements were not mapped to the equation;
er	-	means that the subject erased a correct answer.

Mapped elements. In the Mapping Conditions subjects' responses were coded for the number of elements correctly identified. For example, in the distant condition if a subject has written 7, 10 and **s** , this would be coded as 3 elements. If the subject has written 7, 10 and (**s** + 15), this would be coded as 4 elements, since the **s** is taken as evidence that the subject has called the speed of the other vehicle **s**. Examples of the coding would be:

4e - m	-	4 elements correctly *m*apped to equation;
1e - nm	-	1 element identified but *n*ot *m*apped to equation.

Table/Close Condition

Subject	Overall score	correct solution	correct equation	correct answer	quantity error	matching error	frame error	non-algeb solution	wrong equation	table -> equation
S1	3	1	1	1						f/m
S2	0	0	0	0	1				1	wf/nm
S3	3	1	1	1						f/m
S4	3	1	1	1						f/m
S5	3	1	1	1						f/m
S6	3	1	1	1						f/m
S7	3	1	1	1						f/m
S8	0	0	1	0		1				wf/m
S9	0	0	0	0					1	wf/nm
S10	2	1	1	0						f/m

Table A.2.1. Summary of responses in Table/Close Condition. The Overall Score is calculated by adding1 point for a completely correct equation, 1 for correct answer, and 1 for correct identification of all elements that take part in equation.

Table/Distant condition

Subject	Overall score	correct solution	correct equation	correct answer	quantity error	matching error	frame error	non-algeb solution	wrong equation	table -> equation
S11	0	0	0	0	1		1		1	wf/nm
S12	1	0	0	0						4pf/nm
S13	2	0	1	1	1					wf/m
S14	0	0	0	0			1		1	wf/nm
S15	2	0	1	0	1		1			4f/m
S16	0	0	0	0	1			1?		wf/nm
S17	0	0	0	0	1		1		1	wf/nm
S18	0	0	0	0						wf/nm
S19	2	0	1	0	1					wf/m
S20	2	0	1	0	1					wf/m

Table A.2.2. Summary of responses in Table/Distant Condition.

Mapping/close condition

Subject	Overall score	correct solution	correct equation	correct answer	quantity error	matching error	frame error	non-algeb solution	wrong equation	mapped elements
S21	3	1	1	1						4e-m
S22	3	1	1	1						4e-m
S23	3	1	1	1						4e-m
S24	3	1	1	1						4e-m
S25	2	1	1	0						4e-m
S26	3	1	1	0						4e-m
S27	3	1	1	1						4e-m
S28	3	1	1	1						4e-m
S29	0	0	0	0						1e-m
S30	2	1	1	0						4e-m

Table A.2.3. Summary of results in Mapping/Close Condition.

Mapping/distant condition

Subject	Overall score	correct solution	correct equation	correct answer	quantity error	matching error	frame error	non-algeb solution	wrong equation	mapped elements
S31	3	1	1	1	1					4e-m
S32	3	1	1	1						4e-m
S33	1	0	0	0						4e-nm
S34	2	0	1	0			1			4e-m
S35	0	0	0	0						1e-m
S36	2	1	1	0	1					4e-m
S37	3	1	1	1	1					4e-m
S38	0	0	0	0					1	1e-m
S39	0	0	0	0						-
S40	0	0	0	0						-

Table A.2.4. Summary of results in Mapping/Distant Condition.

A.2.2. Results tables for experiment 3

Table/Distant Condition

Subject	overall score	correct solution	correct equation	correct answer	quantity error	matching error	frame error	non-algeb solution	wrong equation	table -> equation
S1	1	0	0	1	x		x	x	x	pf/nm
S2	0	0	0	1			x	x	x	-
S3	1	0	0	0			x	x?	x	f/nm
S4	1	0	0	0			x		x	f/nm
S5	0	0	0	0	x					wf/nm
S6	2	0	1	1	x					wf/m
S7	0	0	0	1			x	x	x	wf/nm
S8	0	0	0	0	x				x	wf/nm
S9	1	0	0	0		x (t)	x		x	pf/nm
S10	1	0	0	0	x	x (t)				pf/nm
S11	0	0	0	0						f/m -er
S12	1	0	0	0	x		x		x	pf/nm

Table A.2.5. Summary of responses in Table Condition.

Mapping/distant condition

Subject	overall score	correct solution	correct equation	correct answer	quantity error	matching error	non-algeb solution	frame error	wrong equation	mapped elements
S13	3	1	1	1						4e-m
S14	2	0	1	0						4e-m
S15	0	0	0	0		x				1e-m
S16	3	1	1	1						4e-m
S17	3	1	1	1						4e-m
S18	0	0	0	1			x?			3e-m
S19	1	0	0	1	x		x?			4e-m
S20	2	0	1	1	x					3e-m
S21	1	0	0	1	x		x?			4e-m
S22	2	0	1	0						4e-m
S23	2	0	1	0	x					3e-m

Table A.2.6. Summary of results in Mapping Condition.

A.2.3. Subjects' understanding ratings in experiment 3

Table Condition:

Subj	Qu. Overall	d=sxt	goal	3/2	speeds	rep-D	table	Equ.	Sol.	Tot	
S1	5	5	5	5	5	5	4	4	4	42	1
S2	5	5	4	5	5	5	4	5	4.5	42.5	0
S3	4	5	5	4	5	5	5	5	3	41	1
S4	5	5	4	3	5	4	4	4	2.5	36.5	1
S5	4	5	5	4	5	5	4	3	3	38	0
S6	3	5	3	3	5	4	5	2	3	33	2
S7	3	4	3	3	3	3	5	2	2	28	0
S8	4	5	5	4	5	4	3	2	2.5	34.5	0
S9	5	5	5	5	5	5	5	5	4	44	1
S10	5	5	5	4	5	5	5	5	4.5	41.5	1
S11	4	4	5	4	5	5	5	5	4.5	41.5	0
S12	5	5	5	4	5	5	5	5	4.5	43.5	0

Total:	466
Average:	38.83
Mean score:	4.3

Table A.2.7 Summary of 'Understanding' scores for the Table Condition

Mapping Condition:

Subj	Qu. Overall	d=sxt	d1=d2	equat.	map1	diffs.	map2	Equ.	Sol.	Tot	
Score											
S13	4	5	4	4	5	4	5	5	5	41	3
S14	5	5	5	4	5	4	3	4	4	39	2
S15	3	5	4	1	3	4	2	3	4	29	0
S16	4	5	5	5	5	5	4	5	5	43	3
S17	5	5	5	5	5	5	5	5	5	45	3
S18	4	5	3	3	4	3	4	3	3	32	0
S19	3	5	5	4	5	3	4	5	2	36	1
S20	5	5	5	5	5	5	4	5	5	44	2
S21	4	4	4	2	4	3	4	4	2	31	1
S22	5	5	5	4	5	5	5	5	5	44	2
S23	4	5	4	4	5	4	4	4	3	37	2

Total:	421
Average:	38.27
Mean score:	4.2

Table A.2.8. Summary of the 'Understanding' scores for the Mapping condition.

Appendix 3 Think-aloud protocols for subject in SOLO task

A.3.1 Think aloud protocol from S1: Reading the main text of the SOLO training manual

Page 8, under the heading 'Objectives', I'm given under paragraph 2 a list of terms which I am advised I will be able to define and whose relevance I will be able to state to models of cognitive processing. It then goes on to give me various other objectives which I will be able to achieve under paragraph 3 and paragraph 4 which are OK. Paragraph 5 says I will be able to explain why 'representation' is a problem. I would have thought if I was to be discussing representation it should be a term that's included in the list of terms I would be able to define and whose relevance I will be able to state, but it's not.

On various pages I have to keep looking back to find out what the definition of symbol, or symbol structure, or procedure, or basic operation is. Page 15 start of the second line in the second paragraph - I don't know what a 'construct' is. What I mean by that is I've never come across the word in this context before, so I've got to try to figure out from the context exactly what it means therefore my interpretation of it may or may not be correct

OK, this is page 27, and I am typing in NOTE ROVER LIKES FIDO. It's taken me four goes to get it because I didn't notice there were spaces between the end of the first node and dashes, and the beginning of the relation, and then after the relation and before the second node. But I finally got it.

This is still the same page. I am being somewhat scuppered in my understanding here by the fact that page 26 in the middle says 'the actual number of the dashes we type to draw the arrows is unimportant. One or more on each side of the relation name will work', whereas according to the program I'm working, it seems one won't work - it has to be three or else!

Yes, that definitely seems to be the case. I've tried typing it in with one dash, which doesn't work, with three dashes which does work, and with four dashes which doesn't work. So the instructions in my book are contrary to the instructions on this machine.

This is still page 27, and it's SAQ 3. In order to get this FOO BAZ GORT thing, I've put in the FOO BAZ GORT and FOO XLIBRIJK because I don't suppose it's in this program already. In order to get the rest in, I have to type in NOTE space FOO space dash dash dash LOON space dash dash dash, the arrow thing, space FUNG. I missed a space there

before the LOON but there is one. And then I have to do NOTE space FOO space dash dash dash space MXLPLTZ space dash dash dash, a space, GLURTO. I'll just check my answer. *pause* Which seems to be correct as I'd already worked out because the computer has accepted it. Goody. Next page.

Page 28. to end up with ZOG BLORT GRONK and ZOB FI FUM, we would have to type in FORGET ZOG space dash dash dash space GZORN space dash dash dash, a space FOO, and ZOG, sorry, FORGET space ZOG space dash dash dash space BAZ space dash dash dash, a space PLRT.

Page 29. It would have been just slightly handier if the description of which key was used for the arrow head came a couple of pages earlier.

Page 30. I'm omitting SAQ 4 because it doesn't appear to apply, most of it doesn't appear to apply with the particular program of SOLO I'm using.

The question at the bottom of page 30. The second structure is OK because each first node has only one relation name coming from it, whereas in the first structure the first node has the same relation name coming from it twice which isn't allowable.

SAQ 5. You would have to type in FORGET JOHN ISA MAN, FORGET JOHN BREEDS SHEEP, FORGET JOHN MAKE MONEY, and type in JOHN BREEDS ANIMALS, JOHN MAKES PAINTINGS, JOHN HAS PNEUMONIA.

SAQ6 a ABSENT. b ABSENT. c ABSENT dash OOPS - THE PROCEDURE FOR CHECK IS CHECK dash dash etc etc etc. Yes!.

This is page 34. It's a few days since I last did this and I've just tried the CHECK procedure. I've had difficulty because I forgot I had to have a space after the object and before the first set of dashes and after the dashes and... after the relation and before the value.

This is SAQ7. The answers are as follows: a PRESENT MAN, b PRESENT, c ABSENT, d *pause* OOPS - THE FORMATS FOR CHECK ARE etc. Yes, that seems to be right.

I'm on page 38 here, and just for fun I've typed in the TO GRUMBLE instructions on my machine. It doesn't make it particularly clear on the page whether I have to type in the three dots and the colon between numbers one two three and the final instruction DONE. I

didn't put them in and I notice the machine put them in for me. I don't know if that's standard or what I would help if the page said something.

Pages 40 and 41. I hate this automated telephone directory example. I don't think it makes anything clear at all. I can understand the automated telephone directory analogy only because I understand throughout the whole book what I'm required to do, and therefore I can understand the example. But it works that way round and not the other way round, which is presumably the intention.

OK, we're on page 40 now, and I'm reading the instructions about defining procedures and about parameters. I'm probably getting a bit ahead of the instructions here, and I'm going to find out the answers to this question on the next page, but anyway here it is: I've been told, for instance, that to define an INSULT procedure I would type TO INSULT X with the X in slant brackets and then go on and define the INSULT procedure. Page 40 then goes on to say that the program will apply the INSULT procedure to anything within the parameter slot, which seems to make sense, but according to the example it gives at the top of page 40, it will apply it to single names such as HARRY or MARCIA but it won't apply it to FRED, HARRY and SOPHIA. However, there's nothing on the rest of page 40 to explain why the parameter can't be three words plus spaces plus commas. So although I know, for instance, that if I try to apply the DESCRIBE procedure to say FIDO and MARY it won't be accepted because it can only describe one thing at a time. The description of parameters on page 40 doesn't actually make this clear. It might, for instance, have been more straightforward instead of putting an X in the example for the parameter to put in the word NODE because presumably that's going to be all that can ever go within the parameter box. That is one particular node at a time. Although, on reflection, I suppose putting in the word NODE inside the parameter brackets would be slightly confusing too, because presumably it doesn't have to relate only to nodes which are already within the system. Why don't I read page 42 and see if it helps.

Yes I'm right. Half way down page 42 it tells me that parameter names have the same constraints as other names within SOLO. It would have helped really if that had been on page 40, I think.

Page 40 again. Not fair! I just tried to make the think list describe after reading the LIST thing and it won't list DESCRIBE because Ian says that DESCRIBE isn't a procedure like all the other procedures, it's something built into SOLO itself.

SAQ8 on page 44. a SOLO would respond by printing LET ME SAY THAT FIDO IS REALLY GREAT IN FACT I WOULD HAVE TO SAY THAT FIDO IS SUPER. Answer to b:

TO PRAISE /X/ 1 PRINT inverted commas LET ME SAY THAT inverted commas brackets X brackets inverted commas IS REALLY GREAT inverted commas 2 PRINT inverted commas YES inverted commas brackets X brackets inverted commas IS TOO MUCH inverted commas 3 PRINT inverted commas I'M SOLD ON inverted commas X in brackets the X bit (?). Answer to c: if we typed in PRAISE FIDO, SOLO would print LET ME SAY THAT FIDO IS REALLY GREAT, YES, FIDO IS TOO MUCH, I'M SOLD ON FIDO.

Ah! I'm wrong! I didn't realize that the blank spaces after 3 and 1 would eliminate steps 3 and 1, so, I'm wrong.

This is page 45, and it's study activity... study centre activity 3. And I'm having problems with number 1. I've just put in my definition of the PRAISE procedure, and I'm trying to get it listed, and it's saying it can't find the definition of it to list. So I must have done something wrong somewhere like a... space or something, so let's try it all again. OK, got it at last. The problem was I didn't have a space between the PRAISE and the first bracket of the parameter in my definition of the description, and that was because I didn't think there was one in the book on page 44 , which was where I was looking, and I think there may be, it's just not terribly clear.

Right, here we are at SAQ 9 on page 51. The answer to a), JUDGE FIDO, would be FIDO IS A FINE DOG, AND THAT'S ALL I HAVE TO SAY. The answer to b), JUDGE JOHN, is HERE'S MY OPINION OF JOHN, *pause*, JOHN IS ALL RIGHT *pause*, AND THAT'S ALL I HAVE TO SAY. The answer to c) is, *pause*, HERE'S MY OPINION OF MARY, MARY IS VERY THOUGHTFUL, AND THAT'S ALL I HAVE TO SAY. *pause*,. And that seems to be correct according to the back of the book.

This is page 56. STUDY CENTRE ACTIVITY 4 which says 'Define your own procedure called ASSESS which prints out UNHEALTHY if someone either drinks whisky or that person both smokes cigarettes and drinks beer.' The instruction only asks me to have printing out UNHEALTHY if certain conditions are met. But I suppose that means if the conditions aren't met I'm going to have something else printed out, like... HEALTHY. Right. Well. It's got to be something like... TO ASSESS /X/, 1 CHECK /X/ DRINKS WHISKY, 1A IF PRESENT: PRINT 'UNHEALTHY; *pause* EXIT, 1B IF ABSENT: *pause* CONTINUE. *pause*. Now the problem with the having to smoke cigarettes and drinking beer in order to be unhealthy is: I'm going to have to have it continuing on to the next condition after the first, to make sure that I have both conditions met before printing out UNHEALTHY. So 2 has got to be ... what? OK. 2 has got to be CHECK /X/ DRINKS BEER, 2A IF PRESENT CONTINUE *pause* 2 B IF ABSENT PRINT HEALTHY; EXIT, then 3 CHECK X SMOKES CIGARETTES, 3A IF PRESENT PRINT UNHEALTHY...

CONTINUE. No, not CONTINUE. I can have EXIT there, can't I. Then 3B IF ABSENT PRINT HEALTHY *pause* EXIT then DONE. Right.

Right, since that was just made up off the top of my head without actually writing it down and trying it, what I'm going to do now is play all that back what I just said and type it into the computer and see if that works. *pause.* Right. Several minutes later... and, yes, it does work. Good for me. Gold star. Pat on the head.

(Checking program at back of the book the subject discovers that there is a bug in the SOLO program. The next part of the protocol begins five weeks later.)

I'm on pages 56 and 57 which is the hierarchical programming starting. This is where the program went wrong before. So I'm reading it again, and I'll try to see if it can... if it works this time. (*reads*) It seems reasonably straightforward. I should imagine I'm going to have problems with where I'm supposed to have spaces and where I'm not supposed to have spaces, so we'll just try and have a see. I have a hangover today, so I'm not going to be particularly intelligible or articulate, I should think. *pause*

I'm in the middle of shoving in some information to the database so that I can do the exercise on page 58. It's a real nuisance with this program that it doesn't retain what I put into the database.. from the previous time.

Page 58. I've just put in all the things from the database and I've started typing in the JUDGE procedure. I've got to JUDGE /X/ 1 PRINT "HERE'S WHAT I THINK OF" and I can't remember whether I'm supposed to have the inverted commas outside the /X/ which is in slanting brackets or if I'm supposed to have them enclosed within the inverted commas. So I've had to look at the previous page to discover that I'm supposed to have the X in slanting brackets outside the inverted commas.

Yes, of course, I'm half way through it, and of course I reach the problem... which is that I have to have the machine at some stage... distinguish between a male X and a female X so that it can print out either HE or SHE... whatever.

pause

Right, I've typed in my JUDGE procedure and my PRAISE procedure. This is SAQ 10 on page 58. and I'm now trying the examples JUDGE FRED, JUDGE MARY, JUDGE JOHN and JUDGE JOAN. JUDGE FRED and JUDGE MARY have worked properly. JUDGE JOHN hasn't because... um... the... the error message says LINE 2 WITHIN CHECK MUST END

WITH CONTINUE OR EXIT. Which I have to say is something I've forgotten already once but I... I noticed it and remembered to correct it. This is obviously one I've missed so I'm going to have to go back correct it before I see um... whether the thing will carry on working. *pause*

Right, my problem is something which I thought might possibly happen which is that I'm trying to get too many procedures within one step, and I... As far as I can make out I can only have... *pause.* For instance, under 2, if I have a CHECK procedure I can only have 2A and 2B, and my method has got 2A and 2B, and then 2BA and 2BB. Um... I wondered whether this *pause* was correct or not when I was in the middle of defining the JUDGE procedure, but because the machine accepted it and carried on printing 2BA and 2BB I thought it was going to accept it, but it... I don't think it's going to work.

pause

Right, I've altered it now, so that the second subprocedure of line 2 now is 2B CONDEMN /X/, so of course now I'm going to have to... um... define a CONDEMN procedure, which should be reasonably straightforward. *pause*

I'm trying again to JUDGE FRED with my new JUDGE procedure complete with CONDEMN, and of course my previous amendment of line 2 of my JUDGE procedure was wrong because I forgot the dashes, another common mistake. So here we go, yet another EDIT JUDGE procedure.

pause

At last I've had it JUDGE FRED, JUDGE MARY, JUDGE JOHN, and JUDGE JOAN, and I've got the correct answers. My JUDGE procedure is as follows (subject LISTs procedure on screen and reads it out)

```
TO JUDGE /X/
1 PRINT "HERE'S WHAT I THINK OF" /X/
2 CHECK /X/ VOTES INDEPENDENT
2A PRAISE /X/; CONTINUE
2B CONDEMN /X/; CONTINUE
3 PRINT "AND THAT'S ALL I HAVE TO SAY"
DONE
```

My PRAISE PROCEDURE is (subject LISTs procedure on screen and reads it out):

```
TO PRAISE /X/
1 PRINT "I REALLY RESPECT " /X/
2 CHECK /X/ ISA WOMAN
2A PRINT "SHE IS AN INDEPENDENT THINKER"; EXIT
2B PRINT "HE IS AN INDEPENDENT THINKER"; EXIT
DONE
```

And my CONDEMN procedure is... very similar. It is:

```
TO CONDEMN /X/
1 CHECK /X/ ISA WOMAN
1A PRINT "SHE CAN'T THINK FOR HERSELF"; EXIT
1B PRINT "HE CAN'T THINK FOR HIMSELF"; EXIT
DONE
```

pause

I'm now reading the answer to SAQ10 on page 100 which is... pretty... useful. It er... It mentions at the bottom of the page that the control statement EXIT was used in the definitions of PRAISE and CONDEMN, although of course, CONTINUE could also have been used. I did think of that while I was printing in my... definitions. I... I didn't mention it on the tape though. EXIT simply seemed to be... more... logical in terms of how *I* was thinking about it. Although I did appreciate that CONTINUE would work just as well, because it would just... it would... it would simply continue out of the subprocedure back into the main procedure whether you used EXIT or CONTINUE anyway. *pause*. But EXIT seemed to finish the thing off more elegantly, I think.

I've read PROGRESS BOX 5 AND it seems to make sense, although I... it's doubtful whether I could actually,,, explain the concepts of *flow of control* quite as neatly. I mean all the concepts *flow of control , unconditional branching,* and *hierarchical structuring of programs* quite as neatly as the box does. But I think I've grasped the concepts now, um. I'm going to leave it there and continue with page 60 and *making inferences* later. 'Bye.

I'm on page 60 now doing *making inferences,* and I've just gone back to section 4.2 to remind myself about wild card pattern matching which I'd completely forgotten about. *pause* The example in section 4.2 *pause* uses MARY LIKES FRED and uses the wild card symbol in the this node position to check what MARY LIKES. I've been typing it into the computer just because I find that's a much easier way of getting it into my head. So if you put in CHECK MARY --- LIKES ----> ? it says PRESENT FRED. Just for my own interest

and edification I've got CHECK MARY --- ? ---> FRED and it just comes up with the answer PRESENT. It doesn't say PRESENT LIKES or something like that. It does, of course, tell me all that in section 4.2 anyway, but this is just a way of reinforcing my learning of it. Again I'm finding it most annoying that the database I put in yesterday isn't still there today, the computer having been switched off, but there you go.

The information on page 61 doesn't seem to me to be entirely complete, because I think I'll have problems with the particular SOLO program I'm working with. It's explaining that the value of the variable can only be one thing at one time and that 'any value which the variable * might previously have had is discarded to make room for a new value which the asterisk might require as a result of a successful match'. That's fair enough. And it says 'if the match is successful the node which matches the question mark in the given pattern becomes the value of [the] *' That all makes reasonable sense except that the SOLO program I'm working accepts... more than one triple with the same middle node which I don't think it's supposed to, is it? For example, I've just typed in MARY LIKES FRED and MARY LIKES FIDO, you know NOTE MARY --- LIKES ---> FRED, NOTE MARY --- LIKES ---> FIDO; and the machine, of course, has accepted it. But if I now try CHECK MARY --- LIKES ---> ? it only comes back with the answer FRED. Presumably because that's the first one. So the wild card variable thing is only going to go through the information in the database until it finds one thing which matches, the first thing which matches. It's not apparently not going to go through all the information in the database to find everything that matches, it's only going to go for the first one it comes across.

The little picture 'figure 8' on page 61 is as amusing as all the other little pictures in the book but I don't know if it's a particularly brilliant analogy because it tends to suggest, to me at least, that... someone who wasn't understanding this particularly might look at the little box and say 'well, FRED's inside the variable thing therefore the variable's always going to mean FRED'. Whereas of course it means anything that happens to fit whatever the particular query is. So... something like a... a... fishing net, or something, would be better. Sort of flying around and can pick anything out of a whole bunch of possible variables, and only one will fit in the fishing net, or something like that. Shut up, Kathryn.

Page 62, the example right at the top of the page. It gives FRED LIKES MARY and SALLY PLAYS SQUASH 'follow carefully through the definition of SUSS' to see what's going to happen. What's going to... what's going to happen is it's going to say:
HERE'S WHAT FRED LIKES
MARY IS THE ONE

And if you try to SUSS SALLY it's just going to say... em,
HERE'S WHAT SALLY LIKES,
I GIVE UP

This is now SAQ 11 on page 64 . I'm just typing the information into the database SUSAN ISA WOMAN and SUSAN LIKES CARPENTRY. And now I've got to redefine SUSS so that if I type in SUSS SUSAN, SOLO will respond I KNOW SUSAN IS FOND OF CARPENTRY. It says 'The answer to this problem will require you to be clear about the difference between a parameter and a variable.' Well, clearly SUSAN is going to be the parameter that fits in the.. the... em.. X in-slanted-brackets box, and CARPENTRY is going to be the particular variable which is going to fit in the SUSAN LIKES kind of thing. So I'm just going to attempt to type my new definition of SUSS into the computer. It's going to start off with TO SUSS /X/, of course, the X being in the slanted brackets. 1 *pause* There don't seem to be terribly many sort of procedures here so it's going to be
1 CHECK /X/X --- LIKES ---> and then ?thing
1A IF PRESENT PRINT "I KNOW oh no, I've got a space in there after the inverted commas which I don't need OK "I KNOW " /X/ "IS... "IS FOND OF" Now first of all I'm going to have to extend this box or I'm going to run out of space. Right, done that. OK. I KNOW /X/ IS FOND OF, now do I have to have the inverted commas before the asterisk or after the asterisk. I should know this but I'm just checking back on the previous pages. *pause*. Well, outside the inverted commas by the looks of it. I have IS FOND OF" em... asterisk, and if that doesn't work (?) going to tell me anyway, isn't it? IS FOND OF asterisk. Where the heck's the asterisk? Right. OK.
1a IF PRESENT "I KNOW" /X/ 'IS FOND OF" *
1B IF ABSENT em, well it doesn't have to print anything else out, does it, so it's just going to be EXIT. And then DONE. Right. OK. 'I NOW KNOW HOW TO SUSS /X/' Right Let's see if it works. SUSS SUSAN. (types) SUSS SUSAN. 'I KNOW SUSAN IS FOND OF CARPENTRY' but then it says 'OOPS... LINE 1 WITHIN CHECK MUST END WITH CONTINUE OR EXIT' Of course it must. Em... That's interesting, actually, it still comes up with the correct answer, and then tells me about the error message. That's presumably because it's not then exiting from the thing properly. I need a CON... well, I need an EXIT , don't I? at the end of 1A, so I'll just stick that in . EDIT SUSS Em... (types).

Right that was it. The only problem of course about me having to edit anything is that em... I always have so many typing mistakes in it that I've got to abort the thing twice, start again with SOLO from top level, and by the time I've done that I've forgotten what the problem is I'm supposed to be editing in the first place. But I finally got there. We have... em... edited SUSS We have got
CHECK /X/ LIKES ?

IF PRESENT PRINT I KNOW /X/ IS FOND OF; EXIT

1B IF ABSENT; EXIT

DONE

And it SUSSes out SUSAN and says 'I KNOW SUSAN IS FOND OF CARPENTRY' so there we are. SAQ 11. Brilliant. And the answer on page 102 just emphasises the point. Em... the important point being that at step 1a... em 'we use the parameter /x/ to refer to the particular individual to whom we happen to be applying the SUSS procedure, whereas we use the variable * to refer to some entity which SOLO has found as a result of searching through its data base.' Yeah. Fine.

I'm now reading 7.2 on 64, 'Searching through the data base'. SAQ 12 on page 65. The description on page 64 seems reasonably straightforward and most of page 65 is taken up by the database which I haven't yet typed into the computer. I'm just going to try SAQ12 without typing it all in, em, so that I can work out what's happening of course and then I'll (?) how I'm getting on. I'll maybe try typing it in just to see. Anyway. Er.. BOOZETEST FRED. Now we have in the database 'FRED ISA MAN' and 'RED DRINKS BEER' and we have 'BEER ISA BEVERAGE, BEER CONTAINS ALCOHOL', so SOLO's going to do: 1 CHECK FRED DRINKS ?, because FRED of course, is, em, filling in the parameter box /X/, so CHECK FRED DRINKS ? and... and it's going to come up with FRED DRINKS BEER so that's going to satisfy the 1A IF PRESENT: CONTINUE, so it'll then continue on to 2 CHECK BEER. BEER's the variable which has fulfilled the, em, question mark thing in the first CHECK problem. So it's going to have CHECK BEER CONTAINS ALCOHOL and, of course, according to the database, BEER does CONTAIN ALCOHOL so that's then going to satisfy the 2A IF PRESENT CONTINUE thing so it's going to continue again. So it will then going on to 3 and will print AHA FRED IS A REAL BOOZER. Em, b) BOOZETEST MARY. 'MARY ISA WOMAN, MARY DRINKS WINE'. Em, Well it... it has WINE CONTAINS ALCOHOL too in the database, so it is going to print AHA MARY IS A REAL BOOZER. The BOOZETEST SAM. Em, the BOOZETEST MARY one is going to print out the same because basically the same steps will be followed as for BOOZETEST FRED. c) BOOZETEST SAM will go through the following steps: 1 CHECK SAM - SAM being the thing in the parameter box here - CHECK SAM DRINKS ? IF PRESENT: CONTINUE. Now it does say 'SAM DRINKS MILK' so that's going to satisfy the 1A condition and will then continue on to 2 CHECK MILK - MILK being the answer which was arrived at from the question CHECK /X/ DRINKS ? - CHECK MILK CONTAINS ALCOHOL . According to the database MILK doesn't contain ALCOHOL , at least it doesn't say that it does, so that's not going to satisfy the IF PRESENT CONTINUE thing, so it's going to go on to 2B IF ABSENT and then will print NON-ALCOHOLIC and will exit, and that's it done. So the answer to BOOZETEST SAM will be NONALCOHOLIC. And d) BOOZETEST FIDO. The database has 'FIDO ISA DOG, FIDO CHASES CATS.'

So it will be CHECK FIDO DRINKS ? and it won't continue this time because according to the database FIDO doesn't drink anything. So it'll be IF ABSENT: NO INFORMATION. Em so the answer to SAQ 12 will be:

AHA FRED IS A REAL BOOZER

AHA MARY IS A REAL BOOZER

NON-ALCOHOLIC

and NO INFORMATION.

And I don't really think I can be bothered typing all that into the computer just to check that it's right, so I'll just look up the answer at the back. *pause.* And of course it's correct.

Page 66. This is really interesting, this is making inferences.

[protocol ends here]

A.3.2 Think aloud protocol from S1: Reading the appendix of the SOLO training manual

Session 1

S1 reads page 78 Eisenstadt (1978)

E: When you come to do the problems can you talk out loud while you're thinking about it.

S1: Yes I was actually going to do that. Right. Okay.

Reads aloud from page 78

'Given the above description of HARRY, and the most recent definition of INFECT, how would SOLO respond if we typed in SOLO: INFECT HARRY'.

So what it would do is, it would go through to INFECT /X/. Well, in this case HARRY is /X/ so it would go

1 NOTE HARRY HAS FLU

2 CHECK HARRY KISSES ?.

And at that stage it would CHECK HARRY's description to see if he KISSES anyone, and it does have HARRY KISSES JOAN; so JOAN fills in the variable in the question mark and becomes the asterisk in 2A which would be

2A IF PRESENT: NOTE JOAN HAS FLU; EXIT.

Em... it would then EXIT at that point it wouldn't go on to 2B, then DONE. So it would, em... the answer that SOLO would come up with if we typed in INFECT HARRY would be OK JOAN HAS FLU.

That's right isn't it? Yes it does say 'OK' when it's got... when you've NOTEd the thing. It's such a long time since I've looked at this I can't remember what the response is when you ask it to NOTE something. I'm just going to play around with that on this little machine here just to remind myself of, em, of how it responds once you have NOTEs. Em, let's have it in capital letters, shall we?

NOTE JOAN ISA WOMAN: ENTER.

OK JOAN ISA WOMAN.

So the answer to INFECT HARRY would be OK, em... well in fact it would have two things wouldn't it?

Because I suppose it would come up with NOTE HARRY HAS FLU and NOTE JOAN HAS FLU, too, wouldn't it?

So it would have two different things.

I wonder if that actually works. Yes,

E: You're reading page 79.

S1: Yes, I am reading page 79 now.
Yes, em...
My response wasn't taking all the other information that's already in the database, em... like HARRY ISA MAN, HARRY HAS FLU, and things like that, but that's already there.
I'm still reading page 79.
'Step 2 of INFECT essentially captures the following inference "if someone gets the flu, then whomever they kiss gets it as well."'
Middle of page 79.
I'm suddenly being thrown a diagonal line LIZ KISSES JOE KISSES JANE. I don't why they've been drawn diagonally and I don't know if that has any significance particularly. But what it seems to be saying is if we type in INFECT LIZ then, because LIZ KISSES JOE, JOE will end up having flu but it doesn't seem to take another step to have JANE having flu as well because JOE KISSES her.
Page 80 'but what about JANE surely our inference should give her the flu as well!'
reads paragraph 1 on page 80
Yes, in other words only one node can complete the asterisk variable thing so we really need a loop of some sort so it keeps on coming back and adding on and not exiting until it's been through everybody that's been kissed by whatever person has been infected in the chain.
reads
this will be recursion, is it?

E: Could be.

S1: *reads aloud INFECT program on page 80*
In other words it loops.
I don't find the big paragraph right in the middle of page 80 any help at all. Mainly I should think because it doesn't seem strange that a procedure can activate itself. And the whole of that big paragraph starts off 'If it seems somewhat strange that a procedure can activate itself'. It doesn't seem any stranger than the explanation that's given. Generally speaking I haven't found any of the analogies anywhere in this book any use at all. Including the 'passing of a baton from one runner to the other in a relay race' for flow of control.
Okay this little question here in page 81
'Given the latest definition of INFECT, how would SOLO respond if we typed in INFECT LIZ'

We have LIZ KISSES JOE, JOE KISSES JANE, JANE KISSES HENRY, and HENRY SMOKES CIGARS. So we're going to have...

I suppose you want me to go through the entire thing, don't you?

Okay. INFECT LIZ. We start of at the beginning TO INFECT /X/.

Well LIZ is /X/.

So we have NOTE LIZ HAS FLU

CHECK LIZ KISSES ?

Well, LIZ KISSES JOE so

IF PRESENT INFECT JOE,

so then you go right back to the beginning to INFECT JOE. You have NOTE JOE HAS FLU

CHECK JOE KISSES ?

So JOE KISSES JANE.

Em... so again we're into

2A IF PRESENT INFECT the asterisk.

That's the asterisk that fills the question mark in the previous line, so it's going to be

INFECT JANE; EXIT.

Em... TO INFECT JANE, of coursė, you've got to go right back to the beginning again.

TO INFECT JANE

NOTE JANE HAS FLU

CHECK JANE KISSES ?

Well that's HENRY

2A IF PRESENT: INFECT HENRY.

So again we go back to

INFECT HENRY

NOTE HENRY HAS FLU

CHECK HENRY KISSES ?

Well HENRY doesn't actually kiss anybody

so we go to 2B rather than 2A and EXIT

and then DONE.

So then SOLO is going to respond with all the various NOTEs that have been made in the first step.

So we're going to have,

starting off with LIZ,

NOTE LIZ HAS FLU in addition to any other descriptions that exist of LIZ in the data base

but I'm not going to be bothered going back to the previous page,

but it's going to be NOTE LIZ ISA WOMAN or just whatever else LIZ does, it's going to be as well as that NOTE LIZ HAS FLU. And then because of all the various loops that we've gone through we're also going to have NOTE JANE em NOTE JOE HAS FLU in addition to

anything else in JOE's data base and then NOTE JANE HAS FLU.
We're not going...
and NOTE HENRY HAS FLU, and em,
as well as HENRY having flu, of course, he SMOKES CIGARS.
That's obviously going to show up in his description.
So everybody's description in addition to anything they have in them already will have NOTE whoever HAS FLU.
reads on
And that seems to be pretty much the reply that's given on page 81.
I take it I'm getting away with the fact that I'm not listing everything... all the descriptions that exist in everybody's data base?
Because I can't be bothered doing that.

E: It doesn't bother me. It's just to make sure you understand what's going on in the INFECT procedure that's important.

S1: *reads bottom of page 81*
'In fact each particular "expert" received control from some preceding "expert" with the exception of the first one, who received control from us when we us when we originally typed in INFECT LIZ '.
Okay. How long does this go on for?

E: Page 82?

S1: Page 82. Hey, guess what 'the use of a procedure within its own definition [...] is known as recursion.'
(from page 82)
reads page 82
I like recursion, it's good fun.
Sorry, you want me to sit back, I know.

E: It's just so the camera can see where you are.

S1: I take it nobody else is going to be looking at this.

E: I hope not.

S1: God, I don't want anyone else to see my wife, ugh!
Page 82 A.2. 'Eliminating the restriction that FIDO LIKES only one thing at a time:

iteration.' Em, which, of course, I haven't actually been restricted to, because, as far as I can remember, this machine... this program... this... this SOLO thing we have in here will accept as many LIKES as I want to shove in. Can I just test that to remind myself. This has really got nothing to do with what I'm looking at. It's just so I can remember what's going on.

types

What does FIDO like? FIDO LIKES CHUM.

Yes, my memory serves me correctly.

We don't actually have this restriction.

Anyway, Okay.

I'm just staring at this without reading it.

'Given the restriction to one use of the relation LIKES per node, then only one node could possibly match the wild-card symbol (?) at step 2, and this node became the value of the variable *.' So obviously we're going to need some sort of recursion to deal with multiple LIKES on page 83.

Page 84.

The main problem with this is that I'm not really being encouraged to think terribly creatively. Because on page 84 it's telling me... em, a new, a new SUSS thing which... involves a bit of recursion.

And if I were concentrating on this properly and having to actually think creatively I would have thought it out for myself and identified on about page 82, if not about 50 pages earlier, that something like this was going to be necessary.

That's all probably because I'm, well, not quite as interested as I might have been because I haven't got any particular motivation to be desperately interested in it.

Well, like, I haven't.

It's not something that I'm having to use daily.

There's no sort of end goal and therefore I'm finding it perfectly straightforward just to read through everything and understand it perfectly well and deal with it very easily and not actually involving much in the way of active thought.

You know what I mean?

E: What would you... what do you think you'd like to do then? What do you mean by active thought?

S1: Well, em...

Well, like what for instance?

Well, if you're talking in terms of writing this textbook,

it would have been much easier...

well, no, I mean not much easier,

it would have been possible, say, on page...
where does it start describing recursion?
Yes, em before it's...
somewhere anyway before we get to paragraph A.2,
which talks about 'eliminating the restriction that FIDO LIKES only one thing at a time'
it would have been possible to put in some sort of sentence... question to encourage me to think about it. Like, say, something like, can you think of a problem that arises given what you've been told so far?
Because given the information that I've been given so far it should be possible to make a connection about what's necessary to avoid iteration. Because I do have all the information to allow me to arrive at the knowledge that... that a problem exists, and, in fact I would have thought, the knowledge to solve it without having to read page 83, page 84, page 85, and all that as well.

E: What do you understand by iteration so far?

S1: Em.
I don't know. What does it mean?

E: Well, read on. You haven't finished the chapter yet.

S1: No, I know I haven't finished the chapter yet. But what I mean is, it would be easier...
You think this has been a lot of rubbish everything I've said...

E: No.

S1: ... but what I mean is I would learn a lot faster if the thing were encouraging me to, er, to see problems rather than pointing the problems out to me.
I know it's because I'm not thinking about this, because I'm not...

E: I see what you mean.

S1: ...interested enough in it. I mean it could have said something fairly simple in this, this, this page
- that's in the middle of 82 before A.2 -
like 'Can you think of, em, something this might help with.'
And you may be wrong in your answer
but it's still encouraging you to think.

Anyway I've lost where I was now.
I know just above the middle of page 84
It has step 2 FOR EACH CASE OF /X/ LIKES
and I know it's about to go on and explain this FOR EACH CASE OF business.
I'm assuming it's a, a specific sort of tool that's recognized by SOLO like NOTE or DESCRIBE or whatever.
Which I suppose makes this a reasonably high level language, doesn't it? If it can understand things like FOR EACH CASE OF. Yes?

E: (?)

S1: I don't know anything at all about computer languages but I just would have thought that having a computer understand something like FOR EACH CASE OF then it must be fairly sophisticated.

E: Well, it's not really different from things like NOTE.

S1: Well, I suppose not. Okay.
Page 85. We've got the thing 'suppose SOLO's data base contained the following descriptions' and there's the JOHN and MARY bit and then the question how many times the print procedure at sub-step 2A of SUSS would be activated if you're asking it to SUSS JOHN and then SUSS MARY.

E: What are you doing now?

S1: I'm looking back on page 84 to see what the SUSS procedure says. And then looking back on page 85 to see what the question is again. And at substep 2A they're asking me to look at...
So to SUSS /X/ is
1 PRINT "HERE ARE THE THINGS /X/ LIKES
2 FOR EACH CASE OF /X/ LIKES
I like the way the program can step in and out of the inverted commas , the quotation marks, in 1 and isn't...
confused by them in 2.

E: What are you thinking about?

S1: No. I'm not, I'm not... em, I'm not thinking about anything.
That doesn't make sense, any of that.

Never mind. Ignore that statement.
But, of course, the 2 isn't referring to 1
it's referring to the data base.
So that's rubbish. Okay.
Okay, em...
So let's have a look at this question.
SUSS JOHN.
And we have in JOHN's description
JOHN LIKES MUSIC and JOHN LIKES MARY.
So the PRINT procedure at sub-step 2A of the SUSS thing
is going to be activated twice for JOHN.
And it's not going to be activated any times for MARY
because MARY doesn't like anything.
Although the PRINT procedure is going to be activated at a different time, not at sub-step 2A but it's going to activated at sub-step 1 or both.
I can't understand why this SUSS procedure on page 84 hasn't got any EXITs anywhere.
Or any CONTINUE. Surely at 2A it would have to have either EXIT or CONTINUE at the end. Wouldn't it?

E: Why do you think that?

S1: Because I thought you had to have either EXIT or CONTINUE at the end of any substeps.
Is that just my memory failing me?
Just let me look back on this.
See, on page 82, for instance, you have sub-step 2A and 2B, and they both have to be followed by EXIT.
And again on page 80.
I just want to look back, just to see where it gives you substeps.
Because I want to see why it hasn't got them.
(looks back through book)

E: Could you say which pages you are looking at by the way.

S1: Yes, I'm looking at page 49, which has control statements.
I understood from that that I had to have either CONTINUE or EXIT after any sub-step.
So, I'm not terribly sure why there isn't one on sub-step 2 on page 84.
Tell me

E: Can you see a difference between this (?) and what you looked at (?) the control statements?

S1: Well, of course, there isn't a sub-step under 2, there's only one sub-step. I would have thought if there had been any at all there would be a thingy.

E: Are there any other differences?

S1: No. There are none. There are no other differences. Well I mean that is a difference, isn't there? There isn't another sub-step.

E: There isn't another sub-step, no.

S1: And is that why there isn't a CONTINUE or an EXIT?

E: Part of the reason.

S1: Well what's the rest of the reason, then?

E: Em,

S1: I just can't remember this quite well enough, so I'm just going to type this thing in to see exactly what happens when you're putting in this procedure.

E: The SUSS?

S1: Yes this particular one on page 84.
types
What's wrong with that. Oh I haven't got the blasted thingy. What a bloody nuisance.
types
Oh God I can't be bothered doing this

E: You could copy and paste it.

S1: Well that's not going to be any faster than typing it in, is it?

E: No.

S1: *types*
What am I doing?
Right. Two.
So this has obviously got something to do with the FOR EACH CASE OF construct, construct, tool, thing, gimmick, hasn't it?

E: Yeah.

S1: What's it for? That's it going, Okay.
Oh, em, the battery's run out.
The battery probably ran out ten minutes ago.

E: That one's working so we'll just finish this off.

S1: Okay. Em.
types
This is completely unnecessary, isn't it?
Right, well that hasn't got me very far at all.
I still don't know why there doesn't have to be a CONTINUE after 2A, but I have to assume it's because there isn't a 2B and it's got something to do with this FOR EACH CASE OF stuff. Yeah?

E: Yeah, if that's what you think, carry on.

S1: Why don't you just tell me, it would save me a hell of a lot of time. Or is the book going to tell me somewhere?

E: The book ought to have told you.

S1: I know the book ought to have told me, but it hasn't, has it?

E: I don't want to er...

S1: No, no, all right. Okay. Am I being very stupid about this?
At least I noticed. Don't you think that's really brilliant that I noticed there wasn't a CONTINUE there?
You don't care, do you?

E: What are you doing now?

S1: I'm reading the rest of page 85, which is what I said anyway before I started pissing around trying to get this lot in.
God, it really does drag that out, at all. I mean, for half a page it goes on about how it's going to activate the print procedure at sub-step 2A for JOHN twice and none for MARY, which seems so completely basic as to be insulting, and yet it doesn't explain why it doesn't have a CONTINUE at the end of sub-step 2A.
But this is the person who was complaining only ten minutes ago that I wasn't allowed any creative thought.
So at least I've had a wee bit of creative though now, even though I haven't reached any conclusions.
Stop laughing at me.
Right.
So this repeated activation is called iteration.
Now wait a minute. 'This printout may not sound very clever.' Correct. 'But the important point is that the main flow of control, as always, proceeds sequentially step by step in numerical order through the definition of a procedure. Within a step involving FOR EACH, the sub-step is activated *repeatedly* until no more cases of an appropriate pattern-match can be found. This repeated activation is called *iteration*.'
So there you are. I said it was something to do with the FOR EACH procedure, isn't it. It just keeps... Presumably the computer knows, if you've got this FOR EACH procedure that you're using, then the sub-step is just going to keep on being activated repeatedly. So that's it. Isn't it? You don't have to have CONTINUE because the FOR EACH thing sort of has the CONTINUE built in. Which is called iteration! Can I stop now?

E: 8th February, second session.

S1: Poser! Okay We're on page 86 and A.3 FOR EACH WITHIN FOR EACH. So this is sort of like the recursion within iteration. Is that right? Okay. Right, I'm reading page 86 right now. There's a huge temptation when one's sitting in front of this machine to do the whole lot out loud, but I don't suppose that's the best way to do it is it?
Okay. Well, this is brilliant. We're on page 86 now, and it's explaining all the answers to the questions on page 84, which was: why was there only one sub-step, and why doesn't it have a CONTINUE after it.
That was a brief break to take my coffee and take my jersey off.
I'm on page 86. 'Only one substep may be specified within a given usage of FOR EACH' em, it refers me to section 6.6 about the hierarchical structure of programs.
I can't remember exactly what 6.6 says so I'm going to look back and see.

I can't even remember where to find 6.6.

Ah here we are, page 56.

Yes that's quite straightforward then.

It's em, simply saying, 'any given step or sub-step can only specify *one* procedure to be activated. If there's an occasion when you want to specify that *many* things have to be done at a given step or sub-step, then you must first define a new procedure whose definition consists of the many things you want done!'

That's fair enough. Right so back to page 86.

reads

Ah this is interesting; sub-sub-steps which are 1A, 1AA, 1AAA, and I seem to recall this is something I've tried and failed with at some stage earlier in the book. Again it's referring me to the CRAVETEST procedure which is in section 7.4 so I'm just going to go back to that to see what the original one was. So page 7.4, no, page 69, 7.4.

Yep, that's fair enough.

reads page 86.

Back to page 69. Page 70.

I was only looking back there because it says 'remember CRAVETEST was designed to infer that someone craves the contents of whatever they drink' and I couldn't quite remember what the different characteristics of the different drinks were. I know that, em, BOOZE contained ALCOHOL and certainly that was one of the things that was a feature of the CRAVETEST, but I couldn't remember, for instance, MILK containing anything which people craved, and I was really just looking back for interest.

reads page 86.

I'm just rereading this new definition of CRAVETEST several times just to make sure I've grasped exactly what it's doing and how it's going to work. I do remember somewhere earlier in the book it had variables *B and *C. I just can't quite remember exactly how and when they were used. I mean obviously they were used when there's more than one variable.

I do remember somewhere earlier in the book it had variables *B and *C, I just can't quite remember exactly how and when they were used. I mean, obviously, they were used when there is more than one variable. I'm not quite sure why the book is using B and C rather than A. I would have thought A would have been the obvious first one to use. And I can't remember also whether you use * by itself and then *A, *B and *C. So again I'm going to look back in the book to see if I can refresh my memory a bit, because it's a while since I saw this. This is again on page 69 which explains... Yes, it does say that there is *A, *B, *C, up to *Z. It's not anything particularly difficult. It isn't anything that I didn't particularly understand. I think the variables used in this example on page 86 are quite clear. It was just I couldn't understand why B and C were being used instead of * by itself and *A but it really doesn't matter.

reads

Page 87, SAQ 16, which I'm going to read through first of all. Okay, right, I'm going to have to work through this thing in steps to see how I get on and to see if I understand it. Em, my immediate thought, on looking at SAQ 16 without actually trying to answer it, is that the potential problems will be that both MILK and BEER which FRED drinks have more than one contents. MILK CONTAINS CALCIUM and FAT, BEER CONTAINS ALCOHOL and WATER. Em, but anyway let's go through the thing. TO CRAVETEST FRED

FOR EACH CASE OF FRED DRINKS the variable

Now let's just get the order right here.

FOR EACH CASE OF /X/ DRINKS BEER

OKAY, FRED DRINKS... We'll start of with FRED DRINKS BEER, because that 's the first thing that comes up on the data base. FOR EACH CASE OF FRED DRINKS a variable

FOR EACH CASE OF the variable contains another variable

NOTE /X/ CRAVES the other variable. So FOR EACH CASE OF FRED DRINKS BEER

E: If you don't mind, could you move slightly back just a little bit so the camera can see.

S1: Right, we're still in the middle of SAQ 16. And we're still sort of going through it, but at the same time trying to work out how... how it's going to work before I actually try to make it work, which is probably a bit cock-eyed. It would be far more sensible if I just got down and tried it step by step. But I like to figure out first of all where the potential problems are going to be. I... I've already mentioned one problem and the other one is, I don't quite know exactly in which order the thing will work. Because I don't know if it's got to go through the whole test for BEER... Yes, it will of course. Okay. Shut up FOR EACH CASE OF FRED DRINKS thingy. Okay. So it will look at all the data base things and FRED DRINKS BEER will be the first to come out and FOR EACH CASE OF BEER CONTAINS something else NOTE FRED CRAVES whatever the something else is. So, it's going to go through it and it's going to find out that FRED DRINKS BEER because that's the first thing in the data base, and then it's going to say Okay and, in addition to all the rest of the things that are already in the data base for FRED which are listed on page 87 and which I'm not going to go through, it will say, Okay FRED CRAVES em... first of all FRED CRAVES ALCOHOL.

[?]

second BEER thing because that's step... sub-step 1A FOR EACH CASE OF the BEER. It's then going to have: FRED CRAVES WATER and that's it dealt with all the beer stuff, so then it'll go on to FRED DRINKS MILK which is back at step 1 FOR EACH CASE OF /X/ DRINKS thingy, and it'll then have OKAY: FRED CRAVES CALCIUM because that's

the first item listed as contained by MILK and then OKAY FRED CRAVES FAT because that's the second thing that's contained by milk. So it's going to be

OKAY: FRED CRAVES ALCOHOL

OKAY: FRED CRAVES WATER

OKAY: FRED CRAVES CALCIUM

OKAY: FRED CRAVES FAT

and that'll be it. And then it's... Yes, that.. that'll be it done. That's all it prints out. It doesn't print out anything else. Em... I'm now going to look at the answers just to see what I've done. SAQ 16. The answers it gives here OKAY: FRED ISA MAN, FRED DRINKS BEER, FRED DRINKS MILK, which is what's already in the data base as I said, and then

FRED CRAVES ALCOHOL

FRED CRAVES WATER

OKAY: FRED CRAVES CALCIUM

OKAY: FRED CRAVES FAT

which was what I said I think. I'm sure my experimenter will be able to confirm once he goes through the tape that that was the order I said them in.

E: Indeed.

S1: I don't actually know if I need to have this computer on. I'm not testing it on the computer. Em... we've been discussing whether or not we should in fact be typing all this lot and doing it like that, but I've come to the conclusion that that wouldn't be much help because I could quite easily type in the data base on page 87, type in the definition of CRAVETEST and apply it to FRED for /X/, and, although the computer would presumably come up with the right answer, it's not going to show my particular thought processes. It might give me the answers and I could work back from the answers to say why it had listed the items in that particular order, but it's... it em... I don't think it would help at all in trying to establish my thought processes. Although why anybody wants to check my thought processes I don't really know, but... Right, I'm now looking to see how many pages I've got left before I'm finished because... I'm like that. Okay. This is now the bottom of page 87, 'A.4 CHECK node---relation> ? is now obsolete'. Well there you go. Poor old CHECK procedure.

E: For this particular version of SOLO that's actually irrelevant.

S1: Yes, it is. Just out of complete interest, though, for this particular version of SOLO, I want to see that, as far as I can remember, I can still use the CHECK format. I'm sure I've done this before. No, I'm not sure, I think I may have done this before, typed in two nodes in the third position in the triple with the same second position node, and tried a CHECK.

I just want to see what happens because em (?).
types data base on page 87 then
CHECK FRED LIKES ?
SOLO crashes
So has that crashed because of what it's saying here?

E: Not because of what it's saying there, it's because of the way the em... the program works. It doesn't like that at the top level like that.

S1: What top level?

E: Well, going straight in after the SOLO prompt. It would only work in the middle of a program.

S1: Really? Really?

E: Well, if you typed something else like CHECK FRED LIKES WHISKY.

S1: *types CHECK FRED LIKES WHISKY*
I know this has got absolutely nothing to do with anything but it's more fun isn't it?
'PRESENT'.
Okay. That was completely irrelevant.
Yes. 'You may of course still use the format CHECK NODE --- RELATION ---> NODE'.
Which of course we've just established here. 'You may use a CHECK within a FOR EACH,' Okay. Here's a new definition of the old BOOZETEST procedure from section 7.3'.
Right I'm just going to look up 7.3 again. I don't suppose it's necessary at all but I'm going to do it anyway just to remind myself. Em... 7.3 page 67. Right, this is the craving business.
reads example on page 88
That looks nice.

E: What looks nice?

S1: Oh, sorry. It's elegant. I mean rather more elegant than the earlier examples anyway. The way it appears to be making inferences, that's all. I like it. Em.., first of all checking to see if em.. someone drinking something which contains alcohol and if so then... then they crave alcohol then... Oh, NEXT CASE that's a new control statement, That's rather nice.
I'm... I'm reading page 88 now.
How do you write SOLO. I mean how was SOLO written in the first place. I mean how do

you get a machine that doesn't understand SOLO in the first place to recognize SOLO as a language?

E: Well, it's written in Lisp which in turn... LISP is interpreting this, if you like..

S1: Right.

E: and in turn there's a machine code which is interptreting LISP.

S1: Right. I understand that now. Okay. Okay.

reads page 88

reads screen saver on computer screen

'A .44 Magnum beats four aces.' I do hope that's going to be in your protocol. You're not allowed to fudge your results, Ian.

'When you use CHECK within a FOR EACH, the control statement you use after the IF PRESENT and IF ABSENT alternatives must be either EXIT or NEXT CASE.' In other words not CONTINUE. That's fair enough. 'When you use CHECK normally... the control-statements are the usual EXIT or CONTINUE.'

Okay page 89. It's got this little example thing here, BOOZETEST MARTHA and BOOZETEST MARY - according to the data base descriptions contained on page 88.

And this would be what would happen: Okay, BOOZETEST MARTHA. FOR EACH CASE OF MARTHA DRINKS something or other .

Right, so we're going to go through it, and it says..

and first of all MARTHA DRINKS BEER. That's the first thing that's going to come out.

FOR EACH CASE OF MARTHA DRINKS something or other,

BEER is going to lead us on to 1A because MARTHA'S drinking something.

CHECK the *, that's BEER in this first case, CONTAINS ALCOHOL.

So we're going to whip through the data base and we're going to find out that BEER does CONTAIN ALCOHOL.

1AA IF PRESENT: well it is PRESENT. It is fulfilled. So it's going to NOTE /X/ CRAVES ALCOHOL: EXIT.

So we're going to... PRINT. No it's not going to PRINT, it's going to say MARTHA CRAVES ALCOHOL.

And it's going to EXIT there and that'll be it done.

Now is that right? Does it EXIT from the whole thing there? Or does it simply EXIT from 1?

FOR EACH CASE OF /X/... No, it must EXIT from the whole thing, mustn't it?

reads text on page 88

It does say 'EXIT, as always, means "immediately relinquish control from whomever you

got it from."' I don't find that statement particularly helpful. 'When that happens, of course, no other steps would be carried out within this particular procedure.' Yes, okay. If I'd read that properly in the first place I wouldn't have asked the question, would I? But it's useful

(?) because it means I can go back and check.

Okay, so if we're BOOZETESTing MARTHA, we're going to have

MARTHA CRAVES ALCOHOL.

Then I'm going to BOOZETEST MARY. MARY DRINKS MILK

FOR EACH CASE OF MARY DRINKS something it's going to go through CHECK thingy, whatever, CONTAINS ALCOHOL.

So CHECK MILK CONTAINS ALCOHOL, of course we're going to go through the data base and find out that MILK doesn't contain ALCOHOL. So 1AA doesn't work.

1AB 'IF ABSENT PRINT "IS NON-ALCOHOLIC"' is going to work.

Em... So the machine here, in the middle of BOOZETESTING MARY, will print MILK IS NON-ALCOHOLIC then moves on to the NEXT CASE.

This is the NEXT CASE of FOR EACH CASE OF MARY DRINKing whatever.

So if we're going to go on to MARY DRINKS WATER, CHECK WATER CONTAINS ALCOHOL. No, it doesn't.

So 1AA doesn't work, so again we're on 1AB, IF ABSENT PRINT WATER IS NON-ALCOHOLIC; NEXT CASE

and there it's a NEXT CASE so that... we're done, then.

So, for BOOZETEST MARY we're going to have MILK IS NON-ALCOHOLIC WATER IS NON-ALCOHOLIC I GUESS MARY IS NOT A BOOZER.

Okay, is that what happens here? Yep, MARTHA. It's got: OK, MARTHA DRINKS BEER, MARTHA DRINKS MILK, MARTHA CRAVES ALCOHOL.

And it does explain, just in case I was in any doubt at all, which I wasn't actually by this stage, 'immediately after that, the control-statement EXIT says to relinquish control so the BOOZETEST procedure has finished.'

That's immediately before answer (b). And that seems to be okay too.

I'll just read them.. read the rest of page 89 just to make sure I've got the right answers for the right reasons... again.

Page 90. 'A.5 Scope.' I'm reading the page.

This is explaining a little problem so that, em... where you have a variable, it can only be used in the line immediately following... the previous step which was referred to. And that makes perfect sense given this... em... example that's there.

This book is very helpful in that it does things like this. It points out problems before I've thought about them. This is probably something similar to what I was saying earlier on in page 83 -4 sort of thing, when I was complaining about the fact that it wasn't encouraging creative thought.

I still wonder if it might have, sort of , forced the message in slightly more quickly if it had... if it had given me a problem to solve, or at least to attempt to solve which brought out this anomaly. So that I discovered it, as it were, for myself rather than simply having the anomaly pointed out to me. I don't know, anyway... Obviously this... this way works because it's given me this problem and I understand what it's saying.
A restriction of scope. 'A variable may only appear within the *scope* of the FOR EACH construct which originally determines its value.' Well, that seems extremely reasonable, doesn't it?
Right, I'm... I'm still on A.5 and it's this bit which says... which starts off by saying 'related to the issue of "scope" is the fact that when we use the control-statement' etc. That bit there. I'm reading the... I've read it through once. I'm now going to read it through another couple of times till I see exactly what it's meaning. Em... 'it always refers..' 'when we use the control-statement NEXT CASE within a CHECK, it always refers to the most deeply nested FOR EACH construct within which that CHECK procedure immediately appears.'
Yes, that does really make sense. Em... Yes, and, of course, it makes sense if you think in terms of an example anyway.
'A.6 Editing within a sub-step' I am reading this and I'm thinking meanwhile... really I wonder if I were trying to construct... a construct... I got terrible semantics there. If I were trying to define a construct which had a whole nesting of FOR EACH cases, like the example about three quarters of the way down page 90, whether I would actually remember which sub-step I was on. But, of course, SOLO does bring up a sub-step anyway. You don't actually have to remember which sub-step number it is: whether it's 1A, or 1AA or 1AAAB or whatever, because SOLO actually brings up the correct number anyway, doesn't it? So that's a complete waste of thought process, wasn't it?
Page 91 now.
It occurs to me also, at the em... the beginning of page 91, I'm going through this CRAVETEST thing that it's got, and it does occur to me that the book's been pretty good so far about explaining all this FOR EACH CASE OF business, the EACH CASE construct, em... and the CHECKing procedure within the EACH CASE thing, and the NEXT CASE instead of CONTINUE.
That... that's fine. I mean it has all seemed to make perfect sense when I've done it. I just wonder if it mightn't have been more useful again... if it wouldn't have made me understand what was going on a bit better, if I had actually had to design my own construct... define my own construct - let's get the word right - define my own construct which contained... em... each... no, not... no, okay. Sorry, start again.
I wonder if it would have been better to have me define my own procedure, which contained EACH CASE constructs, so that I could really make sure I understood exactly how they worked and that I wasn't going to have any errors in them. I think having to

define my own procedures would make it just that bit more thumped in.

reads page 91

Well that little bit was reasonably straightforward, wasn't it? A.6.

We're now on 'A.7 Iteration versus recursion.'

Before I start reading this, I do recall making some comments somewhere a few pages ago, about what I seemed to think iteration was.

looks back

I can't remember where I said it. It does have it at the bottom of page 85. I've looked back to the bottom of page 85 now. 'Within a step involving FOR EACH, the sub-step is activated *repeatedly* until no more cases of an appropriate pattern-match can be found. This repeated activation is called *iteration*.'

That seems straightforward enough.

I wonder if I can remember what recursion is now.

You see, I... I know I could just go on to page 92, because I know it's going to tell me what recursion is on page 92, but it helps me figure out whether I understand it or not if I can remember for myself what recursion is.

And it's the looping thing rather than the sort of repeating thing.

That sounds really technical, doesn't it?

Again, I'm going to look back.

Looking back to page 78 'Propagating inferences through the data base.' *Reads page 78*

Yes. Okay.

Read page 91

In case it's not clear, I'm still reading A.7 on page 91. I'm not terribly sure these little diagrams are being very useful to me.

It's saying that iteration is fanning out through the network ands it's got this little picture of relations going to node 1, node 2, and all this.

Em... em... and recursion is propagating or going in depth through a sequence of relations.

I can remember obviously what each of the things did through the examples in the book, but I think probably I would find it difficult to explain what iteration meant versus what recursion meant.

'Iteration applies in breadth ... to a fan of nodes all exactly one relation away from the starting node.' Yes, I mean, that's the FOR EACH thing obviously, fair enough, where recursion applies in depth.

So, instead of FOR EACH, you're going for one thing then something different for... from that.

Yes. I think I sort of understand it inside my head but again I would find it very difficult to explain it.

reads page 92

Page 92 now.

That is nice isn't it? This little example here TO INFECT /X/, NOTE /X/ HAS FLU, FOR EACH CASE OF /X/ KISSES thingy, that's the iteration, obviously. Whereas the INFECT in the sub-step 2A is taking you into recursion.

Reads explanation

Yes, that's what it says. 'The FOR EACH construct allows us to "fan out" to discover everyone whom /X/ happens to kiss, while the recursive use of INFECT at sub-step 2A allows us to propagate the "contagious" flu to everyone along a chain of KISSES relations.'

Good example. I mean the KISSES and INFECT things for recursion.

SAQ 17 now.

Quarter to nine. I'm reading it first.

This is nice. Okay. 'Given the latest definition of INFECT, how would SOLO respond if we typed in INFECT JOE'.

Okay. TO INFECT JOE. Okay. TO INFECT /X/, 1 NOTE /X/ HAS FLU. Right, so JOE IS /X/.

It's going to say then ' NOTE JOE HAS FLU. So, SOLO's going to come out with OK, JOE KISSES MARY, JOE KISSES SUE, JOE HAS FLU. Straightforward enough.

Step 2. FOR EACH CASE OF /X/ KISSES wild-card 2A INFECT the variable. Step 2 FOR EACH CASE OF JOE KISSES someone, INFECT that someone.

So it's going to take us first of all to JOE KISSES MARY, which takes us into INFECT MARY. And SOLO is then going to come out with TO INFECT MARY, NOTE... MARY HAS FLU. So it's then going to come out with MARY KISSES FRED, MARY KISSES TOM, MARY HAS FLU. This is where we're really getting into depth here, and how are we going to do it? Em... I mean, which order is it going to come in. Presumably it's still on JOE. Then we have to go through JOE's hierarchy first and then MARY's hierarchy. So, we're still on JOE. We've been through JOE KISSES MARY. We... we're now on JOE KISSES SUE because we're back at sub-step 1 in... in JOE's... JOE's... Okay.

So the first... the first case at step 2 of JOE kissing someone. We're now onto the second case of JOE kissing someone.

So, JOE KISSES SUE. It's going to come out with then, SUE SMOKES... OK, SUE SMOKES CIGARETTES, SUE HAS FLU.

Now, the first case of JOE kissing someone was JOE KISSES MARY. So, MARY has been infected. And it's then going to run through the same process for MARY.

So, FOR EACH CASE OF MARY KISSES someone, we have MARY KISSES FRED. FRED will then be infected at sub-step 2A.

It will come out with OK, FRED SMOKES CIGARETTES, FRED HAS FLU. And the second case of MARY kissing someone will be MARY KISSES TOM. The reply to which is INFECT TOM. So that will be ... SOLO will then say TOM HAS FLEAS, TOM HAS FLU. And... with both MARY... KISSES... the... the recursion will still check to see whether the recipients of MARY's KISSES and in fact the JOE's KISSing of SUE will go on to trigger

another series of infections at a sort of third level, because neither SUE nor FRED nor TOM KISSES anyone. There are no further infections. So that will be the latest result.
I'll now go and look at the answer to that.
I can't actually remember what my rely was to the various order... Right, I don't think I actually have the order right here. In fact, I think I went through JOE one step too soon. I went back to JOE after he kissed MARY. I should in fact have gone through MARY's... the whole of MARY's em... hierarchy first before relinquishing that and going back to JOE.
So, I didn't actually get that right. I think my... my order was wrong. Again that'll be clear from whatever I've said into here.
indicates tape recorder
Let me read through this on page 106 and 107, just to make sure. Yep. Fair enough.
Okay, finished.

A.3.3 SOLO test problems

A.3.3.1. Problem 1

Imagine a rambling four-storey house with a basement. A SOLO database describing part of the house looks like this:

```
ATTIC
  |
  | --- OVER ---> BEDROOM1

BEDROOM1
  |
  | --- ISA ---> ROOM
  |
  | --- OVER ---> LANDING

LANDING
  |
  | --- ISA ---> PLATFORM
  |
  | --- OVER ---> CUPBOARD

CUPBOARD
  |
  | --- OVER ---> LOUNGE

LOUNGE
  |
  | --- ISA ---> ROOM
  |
  | --- OVER ---> KITCHEN

KITCHEN
  |
  |--- ISA ---> ROOM
  |
  | --- OVER ---> BASEMENT

BASEMENT
  |
  | --- HAS ---> CONCRETEFLOOR
```

Now imagine that, after a severe frost, the water tank in the attic bursts.
Write a SOLO program called FLOOD to represent the fact that when the water tank in the attic bursts then the attic and anything below it get flooded.
Once the program has run the items in the database should contain the description overleaf:

```
ATTIC
   |
   | --- OVER ---> BEDROOM1
   |
   | --- IS ---> FLOODED
BEDROOM1
   |
   | --- ISA ---> ROOM
   |
   | --- OVER ---> LANDING
   |
   | --- IS ---> FLOODED
LANDING
   |
   | --- ISA ---> PLATFORM
   |
   | --- OVER ---> CUPBOARD
   |
   | --- IS ---> FLOODED
CUPBOARD
   |
   | --- OVER ---> LOUNGE
   |
   | --- IS ---> FLOODED
LOUNGE
   |
   | --- ISA ---> ROOM
   |
   | --- OVER ---> KITCHEN
   |
   | --- IS ---> FLOODED
KITCHEN
   |
   |--- ISA ---> ROOM
   |
   | --- OVER ---> BASEMENT
   |
   | --- IS ---> FLOODED
BASEMENT
   |
   | --- HAS ---> CONCRETEFLOOR
   |
   | --- IS ---> FLOODED
```

POSSIBLE SOLUTION

```
SOLO: TO FLOOD /X/
...: 1 NOTE /X/ --- IS ---> FLOODED
...: 2 CHECK /X/ --- OVER ---> ?
..... 2A IF PRESENT: FLOOD *; EXIT
..... 2B IF ABSENT: EXIT
..: DONE
```

A.3.3.2 Problem 2

You are an incompetent detective investigating ALEC, a known criminal. Here is a database of the information you have:

```
ALEC
    |
    | --- KNOWS ---> PETE
    |
    | --- KNOWS ---> DAVE
    |
    | --- KNOWS ---> MARY
    |
    | --- KNOWS ---> BILL

PETE
    |
    | --- SELLS ---> CARS
    |
    | --- SELLS ---> VIDEOS
    |
    | --- HAS ---> LOTSOFMONEY

DAVE
    |
    | --- LIKES ---> CATS
    |
    | --- HAS ---> LOTSOFMONEY

MARY
    |
    | --- HAS ---> COMPUTERS
    |
    | --- IS ---> BROKE

BILL
    |
    | --- SELLS ---> VIDEOS
    |
    | --- IS ---> BROKE
```

Write a program called INVESTIGATE which goes through all the people ALEC knows to check whether they sell videos. If they don't, the program should print that they are eliminated from your enquiries. If they do then they should also be checked to see if they have lots of money, if so they should be NOTEd as a suspect, otherwise the program should print that they are eliminated from your enquiries. (See overleaf for what SOLO should print out.)

```
OK...
PETE
        |
        | --- SELLS ---> CARS
        |
        | --- SELLS ---> VIDEOS
        |
        | --- HAS ---> LOTSOFMONEY
        |
        | --- ISA ---> SUSPECT
DAVE  IS ELIMINATED FROM OUR ENQUIRIES
MARY  IS ELIMINATED FROM OUR ENQUIRIES
BILL  IS ELIMINATED FROM OUR ENQUIRIES
```

POSSIBLE SOLUTION

```
SOLO: TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: CHECK * --- SELLS ---> VIDEOS
.......1AA IF PRESENT: IMPLICATE *; NEXT CASE
.......1AB IF ABSENT: PRINT * " IS ELIMINATED FROM OUR ENQUIRIES"; NEXT CASE
...: DONE

OK... I NOW KNOW HOW TO 'INVESTIGATE' /X/

SOLO: TO IMPLICATE /X/
...: 1 CHECK /X/ --- HAS ---> LOTSOFMONEY
.....1A IF PRESENT: NOTE /X/ --- ISA ---> SUSPECT; EXIT
.....1B IF ABSENT: PRINT /X/ " IS ELIMINATED FROM OUR ENQUIRIES"; EXIT
...: DONE

OK... I NOW KNOW HOW TO 'IMPLICATE' /X/
```

A.3.3.3 Problem 3

First of all create a database containing the following information:

```
AVRIL
    |
    | --- LIKES ---> BILL
    |
    | --- LIKES ---> ILONA
    |
    | --- LIKES ---> JOHN

BILL
    |
    | --- LIKES ---> CAROL
    |
    | --- LIKES ---> ISOBEL

JOHN
    |
    | --- LIKES ---> LOTTE

CAROL
    |
    | --- LIKES ---> EMMA
    |
    | --- LIKES ---> FRANK
    |
    |--- LIKES ---> GRAHAM

FRANK
    |
    | --- LIKES ---> GRAHAM

ISOBEL
    |
    | --- LIKES ---> HANNAH
```

Now write a program called 'INVITE' which NOTEs that AVRIL has an invitation to a party and that if AVRIL LIKES someone, BILL for example, then that person also gets invited, as does anybody that BILL likes, and so on. For example:

```
OK...
AVRIL
    |
    | --- LIKES ---> BILL
    |
    | --- LIKES ---> ILONA
    |
    | --- LIKES ---> JOHN
```

```
     |
     | --- LIKES ---> KAREN
     |
     | --- HAS ---> INVITATION

BILL
          |
          | --- LIKES ---> CAROL
          |
          | --- LIKES ---> ISOBEL
          |
          | --- HAS ---> INVITATION
etc
```

POSSIBLE SOLUTION

```
SOLO:  TO INVITE /X/
...: 1 NOTE /X/ --- HAS ---> INVITATION
...: 2 FOR EACH CASE OF /X/ --- LIKES ---> ?
..... 2A: INVITE *
...: DONE
```

A.3.3 Think aloud protocol from S1: Solving the first SOLO problem

E: OK, This is the 12th of February and
this is Problem 1.
S1: *Reads problem*
E: And if you look back over the problem
question could you say which bits you're
looking at.
S1: Yeah. Well, I'm going to read
the whole lot over again.
Reads database out loud.
How do I get to do this?
Do I get to write things down with a pencil
first?
or do you want me to do it all on here?
E: You can if you like, yes. Either way.
S1: Can I not do it both ways?
E: Both ways.
rereads problem
S1: This is recursion
rather than iteration.
If that helps at all.
E: Why do you say that?
S1: Well, because it's not the fan-shaped
thing,
it's the line-shaped thing
on page 91 of the book.
(*S1 gives this page number from memory - she does not refer to the book*)
What is it?
E: What are you looking at?
S1: I'm looking at this again.
I'm looking at the second page (*of problem*)
Well, the first thing that
I'm going to have to do is define a FLOOD
procedure.

Em... and the thing that's sort of causing me
immediate thoughts is:
does my FLOOD procedure actually have to
mention the fact
that the water tank in the attic bursts,
and I suppose it does.
Em. So I'm going to jot down a few notes.
OK. Just the... Just...
the jotting down a few notes is just to sort of
aid my thought processes.
writes
TO FLOOD /X/
Em, well, first of all...
rereads problem
I'm not quite sure exactly
how much depth the program needs me to
have.
I mean does it need me to have somewhere in
the database
the fact that there is a tank in the attic,
for instance?
E: That's the database you start with.
S1: Yeah, that's the database I start with
but, I mean, there's nothing to prevent me
adding things in to the database.
E: No.
S1: Well, my problem is that, at the
moment,
the database doesn't contain the fact
that there is a watertank in the attic,
and therefore it's difficult to have...
I mean, I was thinking, you see,
of starting something like
CHECK WATERTANK IN ATTIC BURSTS.
Well, I mean, obviously it wouldn't say that,
but, I mean, something that would check
that sort of thing.
But since the database at the moment
doesn't actually have the fact that

there is a watertank in the attic,
then, em, that makes it a wee bit difficult,
doesn't it?
It doesn't have it, does it?
That there's a watertank in the attic?
E: It doesn't have it in the database
S1: So does that mean that I have to
add it?
E: You don't have to, you can add it
if you want.
rereads problem
S1: OK, I'm going to look back at my little
book
to see if it can give me any ideas.
E: Which page are you looking at?
S1: I'm not looking at any particular page,
I'm looking at page 90
but not for any particular reason
because there's nothing on it
that's of any particular use at the moment.
Maybe it would help
if I sort of worked backwards.
Maybe it would help
if I read the bit about recursion.
That seems sensible, doesn't it?
OK. I don't have to have the fact...
I'm looking on page 80, em,
which has given a new definition of the
INFECT procedure
so that it keeps on recurring through it.
So I probably...
Right, OK.
It was just to make...
you see, em, I just thought it would be
a bit clearer,
not from the SOLO point of view,
but simply clearer from the point of view
of someone looking at it if...
if the water tank was in it,

but that doesn't really matter an awful lot,
does it?
So it's going to start off with,
em, step1 NOTE ATTIC IS FLOODED
without actually going a step behind that
to explain why the attic is flooded
in the first place.
I think that's the only way I can start really,
isn't it?
OK. Right, this is all just sort of thinking,
We're gong to sort of write it down
and then NOTE ATTIC IS FLOODED
and then we're going to go through it all
and look at it
and see why it's not going to work.
Em, and I'm still looking at page 80 because...
why not?
OK. Then it's going to be now...
reads page 80
now I can see already
that I'm gong to be putting in
CHECK ATTIC is...
OK. So it's not going to be NOTE ATTIC
it's going to be NOTE /X/.
writes
and checks with example in book
I might as well get to there.
Em, right NOTE /X/ IS FLOODED
CHECK /X/ OVER ?
2a IF PRESENT...
IF PRESENT FLOOD *.
writes
Now this little thing I'm jotting down
initially
is simply copying what's on page 80
for the new INFECT procedure
then I'm going to have to read it
and discover that I've got to, sort of...
well anyway...

writes

Right so far what I've written down is

TO FLOOD /X/

1 NOTE /X/ IS FLOODED

2 CHECK /X/ OVER ?

2a IF PRESENT: FLOOD *; EXIT

2B IF ABSENT: EXIT

DONE

Now I just simply can't remember from the
last time I did this
whether that will take us right through
everything,
or whether that will simply take us
through one case of it;
so I'm just going to read my book
a wee bit further on.

reads

Anyway probably the best thing to do
is type the whole lot in to start with,
isn't it?
To actually put the database in
so that I can just see what happens.
Em, and I'm annoyed that I can't remember
if that takes us right through everything
or just...

reads

E: Which page is this?

S1: This is page 82.

skims quickly over pages 82, 83

And I'm looking on to the iteration,
but that doesn't matter.
I've decided that doesn't matter
because we're not doing a fan thing here
we're doing a line thing,
but I may be wrong.
In fact I may be seriously wrong.

reads p92.

In fact, I'm now looking at page 92
and I suspect that

what I want is something rather more
like the INFECT thing on there
which is NOTE /X/ IS FLOODED
refers continually to example
and then have a
FOR EACH CASE OF /X/ OVER ?
INFECT *.
Right that's probably what I need,
isn't it?
Right, scrap...
Em. OK. FLOOD procedure mark II.
writes
TO FLOOD /X/
E: Why don't write your database into
the computer then you can check it as you write.
S1: Right. OK. That sounds fair enough.
Types database into computer
E: Arrowhead.
S1: Arrowhead.
types
NOTE LOUNGE ISA ROOM.
types
So do you want me to check the one
I've just scrubbed out?
E: That's up to you.
S1: Is it?
types
TO FLOOD /X/
Right, I'm thinking. I'm thinking.
E: Can you say what it is you're thinking?
S1: Yes. Yes, I mean I'm...
I'm sort of looking...
I'm... I'm just basically, as you've gathered,
copying the examples in the book,
and the one on page 92, em... is...
E: What are you looking at now?
S1: I'm looking at the first page of
the problem
just to see exactly what the question's asking

reads:
"If the watertank in the attic bursts and
everything below it gets flooded"
So the attic itself doesn't have to get
flooded?
Is that correct?
Or does the attic too get flooded?
Well, I mean, it... it implies
that the rooms below it get flooded
but the attic doesn't necessarily.
E: You can assume the attic gets flooded
as well.
S1: Right. OK. I'm going to try my first
problem and see what happens.
This is because I really haven't a clue.
types:
...: 2 CHECK /X/ --- OVER ---> ?
No. no this won't do.
I'm typing in the first thing.
E: Could you say what it is you're typing?
S1: Yes. Yes, I'm typing the problem
that I wrote down which is
CHECK /X/ IS OVER something or other.
E: You've missed a step...
(from the subject's written version of problem)
S1: Yes. Yes I know. I know I have.
So I have.
That's what makes things entirely
different, in fact.
Em... NOTE /X/ IS FLOODED.
Back. Back to... what page was I on,
page 82 or something?
Page 80, I'm on page 80.
reads
Yeah, OK I think the first example
is probably going to work, yes.
OK.
types in program she has already written:
Do I need a space there?

Yes, I do, don't I?

Where's the asterisk?

types

[*The program now is:*

TO FLOOD /X/

...1 NOTE /X/ — IS --> FLOODED

.....2A IF PRESENT: FLOOD *; EXIT

.....2B IF ABSENT: EXIT

...: DONE]

[*SOLO responds*

OK... I NOW KNOW HOW TO 'FLOOD' /X/

SOLO:]

S1: OK

types:

FLOOD ATTIC

[*SOLO responds:*

OK...
ATTIC
|
| --- OVER ---> BEDROOM
|
| --- IS ---> FLOODED

OK...
BEDROOM
|
| --- ISA ---> ROOM
|
| --- OVER ---> LANDING
|
| --- IS ---> FLOODED

OK...
LANDING
|
| --- ISA ---> PLATFORM
|
| --- OVER ---> CUPBOARD
|
| --- IS ---> FLOODED

OK...
CUPBOARD
|
| --- OVER ---> LOUNGE

```
      |
      | --- IS ---> FLOODED

OK...
   LOUNGE
      |
      | --- ISA ---> ROOM
      |
      | --- OVER ---> KITCHEN
      |
      | --- IS ---> FLOODED
```

```
OK...
   KITCHEN
      |
      | --- ISA ---> ROOM
      |
      | --- OVER ---> BASEMENT
      |
      | --- IS ---> FLOODED

OK...
   BASEMENT
      |
      | --- HAS ---> CONCRETEFLOOR
      |
      | --- IS ---> FLOODED]
```

S1: That looks not bad, eh? *evaluate solution*

Is that right? *query: checking*

E: That's right.

A.3.4. Think aloud protocol from S1: Solving the second SOLO problem

E: This is problem 2.

S1: This is still the 12th of February.

(reads problem aloud)

'You are an incompetent detective
investigating ALEC a known criminal. Here is
a database of the information you have.'

Right, this time I'm actually going
to type this lot straight off into the database
and that's going to have it in the database,
and I'll be reading it at the same time,
which seems to be intelligent, doesn't it?

(reads database aloud as it is typed)

Right, we've got a database.

OK, problem.

(reads problem aloud)

OK. Right. I'm just reading the problem again
and thinking.

(re-reads problem)

OK, so we're going to have to start with a
FOR EACH CASE OF ALEC KNOWS
somebody or other.

I'm just going to check in the book, em,
to make sure I've got the exact wording
of FOR EACH.

I think it's FOR EACH CASE OF /X/,
but I just want to make sure that's exactly
what the wording of the thing is.

Em, if I can find it.

I'm looking for the right page now.

That's the blasted answers.

reads page 91 CRAVETEST example

'FOR EACH CASE OF' yes,
it says 'FOR EACH CASE OF'.

(reads)

Now, what am I going to do?

We want to start off with

TO INVESTIGATE /X/; 1
and presumably what we want to do here
is investigate ALEC,
yes. Em...
So we're going to start of with
FOR EACH CASE OF ALEC KNOWS
whatshisname
(types)
That's not actually what I'm going to type in.
Not word for word.
FOR EACH CASE OF /X/.
Now, how do I do this?
Dot, dot, dot, I suppose.
I'm looking at page 91
because it's got an example of one of these,
em, FOR EACH CASE OF.
Right, FOR EACH CASE OF /X/ KNOWS.
Now what do we need?
A question mark.
Now I'm thinking...
on page 91 I'm looking at the CRAVETEST
thing in the middle
which is just simply the first example I
came across,
which had the an example of the FOR EACH
thing in it,
because, as you recall, I was looking to see
what the exact wording of the FOR EACH
construct was.
Or was it a procedure?
I can't remember.
I should know that.
Was it a procedure or a construct?
The INVESTIGATE is a procedure
so the FOR EACH is a construct, yeah?
E: That'll do, yeah. FOR EACH is a
construct.
S1: Does it matter, anyway?
No. Right.

So I've got some idea.
Anyway, now I'm just sort of looking
because on page 91 the particular example
I'm looking at
has got ?B and ?Q and stuff like that.
So I'm still thinking:
well do I actually need, well, just a
question mark?
Or do I need a question mark
something-or-other else?
Em, And this serves me right for
just dipping into the book
and looking at the first possible example of it
that I come across
without actually reading what I'm doing,
doesn't it?
So why don't I go back a little bit?
Well, I mean, the whole idea about page 91 is
that it's just shoving...
shoving the different letter...
letters in after the question mark
to simply demonstrate to you
how the thing can be edited anyway.
(reads page 91)
1B CHECK *.
Now what am I doing?
Back.
I'm looking back at the question.
OK, 'Go through all the people ALEC
knows to check whether they sell videos',
OK, CHECK * SELLS...
now, hang on, what do I need?
Dash thingy, don't I?
Dash, dash, dash, SELLS VIDEOS.
Now... I just want to remind myself
how I continue after I've had a CHECK
procedure within a for EACH CASE OF thing.
(reads - finds page 88)
I'm looking back to page 88,

and it's just going to come up...
(reads BOOZETEST example)
So it's... It's just going to come up
1AA IF PRESENT:
OK, now, what's this going to do?
It's going to...
Right let me just think about this.
If I continue...
reads screen
CHECK * SELLS VIDEOS; IF PRESENT...
I can't remember if I can have a CHECK
procedure within a CHECK procedure.
Can I?
You're not going to tell me are you?
(reads)
This is trying to remember
all the embedded steps.
Yes. em, no, it's not.
OK, I'm still thinking about this.
reads screen
OK. CHECK * SELLS VIDEOS;
1AA IF PRESENT...
Certainly 1AB is going to be IF ABSENT
and then I've got to print... em, you know,
reads problem
'they are eliminated from the enquiries'
sort of bit.
Em, so that's...
the ABSENT is actually straightforward
enough.
If it's PRESENT, and they do sell videos,
I've then got to go on and CHECK
whether or not they have lots of money.
It's just quite how I get into that.
reads screen
Anyway. At the moment the only thing
I can think of is having another CHECK
procedure in here too.
I just thought I couldn't do that for some

reason,
but I can't remember it.
flicks through book
The answer is in here somewhere
but I've no idea how far back it is,
which is a right nuisance.
I'm looking all over the place
because I've got no idea where I should be
looking .
I'm on page 90 now.
Aha! I want a NEXT CASE thing, perhaps.
Do I?
reads text on page 90
Right, I'm on page 90 'related to the
issue of "scope" is the fact that when we use
the control-statement NEXT CASE within a
CHECK, it always refers to the most deeply
"nested" FOR EACH construct within which
that CHECK procedure immediately appears.'
Now, is that going to work in here?
reads screen
OK. FOR EACH CASE OF /X/ KNOWS...
CHECK /X/ SELLS VIDEOS; IF PRESENT...
Now is that right?
Because if I put in NEXT CASE
it's gong to take me right back to...
reads text on page 90
the FOR EACH thing.
(reads)
E: This is page 90 you're reading.
S1: Yes, I said that. 89. 88. I haven't read 89 yet.
E: What is it you're looking at on 88 is it the,
em, example or the text?
S1: I'm looking at the example,
the BOOZETEST example on page 88.
But it's not going quite far enough,
that example on page 88,
because, em, it's only got one variable,
which is the alcohol,

the... the thingy contains alcohol,
whereas I've got two,
which is both that they sell videos
and that they have lots of money.
OK, FOR EACH CASE OF /X/ KNOWS
thingy;
CHECK /X/ SELLS VIDEOS;
1AA IF PRESENT...
I can't remember if this works.
OK, em, because I have in my head at the
moment this, sort of...
that I can get round it by having another
CHECK thingy here,
which I can't remember if I can actually do
or not,
and I can't remember where to find the
answer,
although I'm sure it's in here somewhere,
I'm just going to try it and see what happens
and it's probably going to tell me I can't.
No. Anyway.
Because what's the variable going to be called?
Is it going to be called just asterisk again?
reads screen
I'm going on.
I'm reading page 92 the INFECT thingy here
because it tells me how to do...
or it tells me...
it gives an example of iteration and
recursion being used together.
(reads INFECT example
So...
reads screen
E: What... Are you just reading through the
thing on the screen at the moment?
S1: Yes. At least I'm staring at it blankly.
(laughs)
I'm still... I'm looking back at the book again
because I still want to see what I can do with

this, em...
flicks backward and forward through book
CHECK thing within a CHECK thing
because I know I can't do it
but I just want to see somewhere it tells me
I can't do it.
What I really need is some sort of procedure
that will do wee loops.
E: What do you mean by loops?
S1: Don't know.
(types)
reads screen
reads problem
Oh, now I'm trying to remember
where I 've got to have the blasted s
semi-colons here.
Is that where I have to have the semi-colons,
the... I mean the inverted commas?
Or do I have to have them before the /X/?
E: Anything you want printed that's not a
variable should be in inverted commas.
S1: Yeah, I know that
but the variable /X/ doesn't go within the
inverted commas,
that's what I mean, does it?
E: Yeah.
S1: Yeah it doesn't?
E: It doesn't.
S1: Yes, it doesn't. OK, thank you. Em,
(E tries to adjust window on screen. Some of the S's code has overflowed to the next line causing SOLO to abort.)
OK. 1 FOR EACH CASE OF thingy.
Right, hang on. Think about this.
FOR EACH CASE OF somebody KNOWS
somebody else
CHECK that somebody sells VIDEOS;
IF PRESENT: CONTINUE; IF ABSENT...
[S1's program so far is

SOLO: TO INVESTIGATE /X/

...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?

.....1A: CHECK * --- SELLS ---> VIDEOS

.....1AA IF PRESENT: CONTINUE

.....1AB IF ABSENT: PRINT /X/ "IS ELIMINATED FROM OUR ENQUIRIES"; EXIT]

OK. Right 1B

I can't do that can I?

I can't have a 1B within a FOR EACH

construct, can I?

I want two CHECK procedures within the

FOR EACH thing

and I can only have a IA, can't I?

E: Yes.

S1: It's not fair.

(reads screen)

No, got it.

Right, what I'm going to do here

is type DONE

just so I can get out of this one I'm typing in here

and start the whole thing again.

Do I have to go in through an

EDIT INVESTIGATE

or can I just start off and type

TO INVESTIGATE again?

E: Start TO INVESTIGATE again, it'll

probably be quicker.

S1: Yes.

(types)

I've probably forgotten now

what I was going to do.

(types)

[first program now looks like this:

SOLO: TO INVESTIGATE /X/

...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?

.....1A: CHECK * --- SELLS ---> VIDEOS

.....1AA IF PRESENT: CONTINUE

.....1AB IF ABSENT: PRINT /X/ "IS ELIMINATED FROM OUR ENQUIRIES"; EXIT

...: DONE

OK... I NOW KNOW HOW TO 'INVESTIGATE' /X/

S1 retypes:

SOLO: TO INVESTIGATE /X/

...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?

.....1A:]

Right I'm going to try something else here.

And the problem I'm now going to have

is that I'm probably going to start having

too many variables.

I'm going to have to call them different things

like ?B and stuff like that

and it's not going to help if I don't put all the

letters in either, does it?

(corrects what she has typed).

And that's where my problems are going to

arise.

But that, at least, is something I can probably

sort out.

FOR EACH CASE OF /X/ KNOWS thingy

FOR EACH CASE... -

(types)

and I don't know if this is will work -

OF * being the answer to the question mark...

OK. FOR EACH CASE OF * SELLS...

OK this is getting too late at night.

EACH CASE OF * SELLS VIDEOS

Right 1A CHECK...

Now, this is where I'm going to have to start

putting in different letters, isn't it?

CHECK * A HAS LOTSOFMONEY.

That's what I'm going to need now, isn't it?

CHECK the asterisk A.

Or is that not right?

Well, why don't I...

E: That's a different variable?

S1: Yeah, well that's what I...

Yes, do I need a different variable?

CHECK...

OK. * HAS...

[*types*:

1AA: CHECK * --- HAS --->
LOTSOFMONEY])
Right now I've got to have a
NEXT CASE here,
don't I?
1AA IF PRESENT...
My problem is the numbers of my variables,
isn't it?
flicks through book
Here I... I mean.
indicates asterisk and question mark
How many A's or whatever I need after them
all.
Or if I ever need a variable after them all,
or if they're all the same variable.
I know I should be able to work this out.
NOTE *
[*types*:
NOTE * --- ISA ---> SUSPECT]
reads screen
I'm not terribly sure
if I can do substep 1A after substep 1
and the whole thing to make sense
but, never mind we'll see, won't we?
and if I can't finish this tonight
then I'll go to bed and think about it all day
tomorrow.
IF PRESENT NOTE thingy ISA SUSPECT.
Now I suppose...
I can't remember...
I just can't think any more.

E: I think I should point out, you've got this
in the wrong place.
S1: What the NEXT CASE?
E: You don't have that immediately after
the CHECK.
[S1 has written:
.....1AA: CHECK * --- HAS --->
LOTSOFMONEY; NEXT CASE

E assumes that S1 has made a slip]

S1: Oh, I've got it...

I've got it after now, haven't...

yes, that's right.

OK, em, I can't get rid of all that now, can I?

E: You could try.

[S1 deletes the NEXT CASE from line 1AA]

S1: I don't think that works though.

E: No I don't think so either.

S1: Once you've put it in it's in.

Em, so... Why's it doing that?

E: You haven't got (?)

S1: Oh aye right.

Can I not just put DONE now.

Em, this isn't going to work is it?

I'm just going to put EXIT and DONE now

and then mend that

because it's not going to accept it.

I know it's not.

[program now is:

SOLO: TO INVESTIGATE /X/

...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?

.....1A: FOR EACH CASE OF * --- SELLS ---> VIDEOS

.....1AA: CHECK * --- HAS ---> LOTSOFMONEY

.....1AAA IF PRESENT: NOTE * --- ISA ---> SUSPECT; NEXT CASE

.....1AAB IF ABSENT: EXIT

...: DONE]

OK here we go again. Mark III.

(types)

I'm just looking to see about this NEXT CASE

thing.

The little explanation it gives you.

(reads middle page 88)

OK.

[*types*:

SOLO: TO INVESTIGATE /X/

...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?

.....1A: FOR EACH CASE OF * --- SELLS ---> VIDEOS

.....1AA: CHECK * --- HAS ---> LOTSOFMONEY

.....1AAA IF PRESENT: NOTE * --- ISA ---> SUSPECT; NEXT CASE

.....1AAB IF ABSENT: PRINT /X/ "IS ELIMINATED FROM OUR ENQUIRIES"; NEXT

...: CASE

S1: Is that as far as I can do?

Will it not work, then?

(line 1AAB has spilled over to the next line

which will cause SOLO to abort.

E: Try it. Hit the ENTER and see.

[SOLO responds:

;;; ERROR: OOPS... FOUND CASE INSTEAD

OF A LINE NUMBER OR DONE]

S1: What do I do?

Which one do I press?

E: Abort.

S1: Now, what do I do now?

Em, start again.

Do have to start all that again?

Can you copy all that for me please?

E: Yeah.

S1: Well, I mean we can have the

'FROM OUR ENQUIRIES' bit out

if necessary.

You can just have /X/ 'IS ELIMINATED'.

/X/ IS EXTERMINATED as far as I'm

concerned.

[*E copies and pastes up to 1AAB in the last code*]

S1: I'll be lost now.

(There is some trouble with SOLO which

aborts since ENTER was accidentally

pressed twice)

S1: So I can't remember where I am now.

This isn't fair.

E: I think you're on 2 or DONE or

whichever you are on to.

[*program now reads:*

SOLO: TO INVESTIGATE /X/

...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?

.....1A: FOR EACH CASE OF * --- SELLS ---> VIDEOS

.....1AA: CHECK * --- HAS ---> LOTSOFMONEY

.....1AAA IF PRESENT: NOTE * --- ISA ---> SUSPECT; NEXT CASE

.....1AAB IF ABSENT: PRINT /X/ "IS ELIMINATED"; NEXT CASE

S1: FOR EACH CASE OF /X/ KNOWS

something or other.

I can't remember what the problem is.

(reads problem).

Right, wait a minute.

FOR EACH CASE OF /X/ KNOWS

(reads through what has been written on screen)

because it's also got to eliminate things

from the enquiries

if they don't sell videos.

At the moment I haven't got anything to

check to see if it sells videos.

Em, because I want something to print...

if it doesn't sell videos

I want something to print that it's also

eliminated...

also to print that it's eliminated from the

enquiries and...

at the moment it's not going to do that

because what the thing's prompting me to put

in now is 2

Hang on, let's see what we've got here.

See ALEC actually knows all the people

DAVE, MARY and BILL.

And at the moment I can't see what I can do to

print that somebody's eliminated

if they don't sell videos.

Because the 2 that I'm putting in at the

moment

isn't within the FOR EACH CASE in step 1.

Have you done this?

You've solved it so it is possible?

E: Yes.

S1: (*reads screen*)

I'm going to look back at my little book here

to see.

E: Page 86.

S1: So it is.

E: Which part are you reading? All of it?

S1: Yes. Yes, this must...

reads text

Right, 'only substep may be specified within a given usage of FOR EACH. This means that if you wanted to have, say, five different procedures activated within a given substep', I'd have 'to define a single new procedure which itself specified the activation of all of these five procedures as part of its own definition.'

That's what I really have to do.

So let's just...

E: Do you want to try to see what happens if you try out your program or are you going to forget about it entirely and do a different one?

S1: Oh, yes OK. I'll try it if you like.

types:

SOLO: INVESTIGATE PETE

[SOLO responds:

```
OK...
   PETE
     |
     | --- SELLS ---> CARS
     |
     | --- SELLS ---> VIDEOS
     |
     | --- HAS ---> LOTSOFMONEY
     |
     | --- ISA ---> SUSPECT

OOPS... I DONT HAVE A PROCEDURE CALLED
'NIL'
```

S1: Where did this NIL come from?

E: I haven't the faintest idea.

S1: I've got half way there so far,

reads screen

but as I said I didn't have the, em...

What is this, what is this I mean what is

this?

(*referring to the NIL*)

E: I don't know it must have been the double

hit of the ENTER button that did that.

S1: Yes.

E: NIL is what LISP calls an empty list.

S!: Yes, but anyway, this is exactly what

I thought was going to happen

because I didn't...

I wasn't able to get into the, em...

I couldn't include the PRINT ELIMINATE

thing FROM MY ENQUIRIES as a next step.

reads screen

Right, well, OK, I mean, I can...

I can see ways round this

but I think they might...

may be going without...

outwith the, em, the limits of the question,

but I'm going to do them anyway, so there.

E: What, er, do you mean by that?

What ways do you see?

S1: Ah well, you're just about to find out,

aren't you?

Well, it involves having a different procedure

within the INVESTIGATE procedure.

E: Do you want to save time by cutting and

pasting from the previous procedure or are

you just going to start from scratch?

S1: I'm not very sure.

No, I'm starting from scratch.

Because I'm going to have to start

by defining my other procedure

that I'm going to have in the middle of it.

Am I allowed to do that within the

(?) of the question?

E: Yes. You don't have to do it first,

you can do it second it doesn't matter which
order they're in.
S1: Yes, that's right.
Oh well, OK, em.
(*types :*
TO INVESTIGATE /X/
1 FOR EACH CASE OF /X/ --- KNOWS ---> ?]
But the way I'm doing it isn't going to have
the whole lot within one definition,
is that all right?
E: Uhuh.
S1: Oh, I see, right. OK.
TYPES:
CHECK /X/ --- IS ---> DODGY
E: CHECK /X/ IS DODGY?
S1: * IS DODGY.
*replaces /X/ with **
And then I'm going to have to define a
procedure called DODGY.
E: But that (*referring to screen*) is not a
procedure.
S1: Not yet. I can design another a little
program.
E: I can see your point but that can't be a
procedure. Sorry, carry on, do what you want.
S1: Well, what I was going to do was you see,
was after I finish the INVESTIGATE thingy,
is define another procedure
which results in /X/ being DODGY
if /X/ both sells videos and has lots of money.
E: OK
S1: Will that not work?
E: Yes, there's no reason why that
shouldn't work.
E: But that's not what you wanted me to do?
E: You can do it anyway you like.
That will work.
S1: But it's not the neatest way of doing it,
is it?

E: It probably is.

S1: But it's probably not. It's not neat.

I mean, it's not elegant, is it?

(*reads screen*)

But you don't like it that way, do you?

E: I do like it that way, yes.

S1: But it's not a good way.

E: Can you think of a better one?

S1: Not right at the moment,

because I'm in the middle of thinking about

this one.

E: Do this one.

S1: *types*:

SOLO: TO INVESTIGATE /X/

...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?

.....1A: CHECK * --- IS ---> DODGY

.....1AA IF PRESENT: NOTE * --- ISA ---> SUSPECT; NEXT CASE

.....1AB IF ABSENT: PRINT /X/ "IS ELIMINATED"; NEXT CASE

S1: Am I right in having that thing there?

E: What? The /X/?

S1: Yes.

E: Well, what would the /X/ refer back to?

S1: Well... so it should be an asterisk, then.

Yes. Shouldn't it?

Let me just check on that.

flicks through book

(*reads screen*)

I'm still checking my NEXT CASEs things.

flicks through book

I can't remember what page it's on.

Oh yes, 88.

Where are we?

reads screen

types

E: It's just DONE.

S1: Ah. Em, will that not all work then.

E: I don't know try it. You could edit it and

just...

S1: and just exit to the 2.

E: Get the 2 out.

S1: types

[*program is now*:

```
SOLO: TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: CHECK * --- IS ---> DODGY
.......1AA IF PRESENT: NOTE * --- ISA ---> SUSPECT; NEXT CASE
.......1AB IF ABSENT: PRINT * "IS ELIMINATED"; NEXT CASE
...: 2 DONE
...: DONE
OK... I NOW KNOW HOW TO 'INVESTIGATE' /X/
SOLO: EDIT INVESTIGATE
...: 2
...: DONE
OK... I HAVE RE-DEFINED HOW TO 'INVESTIGATE' /X/]
```

Now.

E: So your plan is to have this procedure

called INVESTIGATE and another one.

S1: Another one.

Which results in putting this...

I mean...

E: You'd run it first, would you?

S1: No, em no.

I'm just sort of thinking through all this.

I don't actually, em...

Yes, obviously, I would...

I mean, obviously I would sort of

run the DODGY thing first.

But I don't actually want to do that.

Because I want the whole thing to simply

to run with what's in the database.

At the moment.

So, really...

reads her solution

Right, we'll do it again.

E: The INVESTIGATE?

S1: Yeah, but I want to have...

I've figured out what sort of thing I want in

there (indicating DODGY)
and, I mean, you're right
that's not the sort of thing.
I want another procedure in there, don't I?
Em... something like
something called SUSS /X/
where the procedure called SUSS /X/
checks to see whether /X/ was both selling
videos
and having lots of money
and if...
E: Uhuh, Right.
S1: So was some sort of thing...
So, em, oh God!
E: You've been at it for over an hour.
Do you want to stop for a bit?
S1: No I want to finish it. Stop for a bit?
Stop for a bit.
S1 has half hour break
S1: I'm looking through the iteration bit
on page 82 and 83 to try and remember it.
The problem I'm having her is that,
although on page 83 82 and 83 the book
tells me how to do iteration there,
the first node and the third node on the triple
are related by the same second node
whereas in this case
what I'm trying to do is something where
the second nodes are also varying.
At least I've got two variable lines going down
instead of just the one.
I'll draw a diagram of what I'm trying to do.
The ordinary iteration is this fanning out thing
and the recursion is this thing here.
What I'm trying to do
what the problem is...
is first of all I'm doing the iteration thing
for the FOR EACH
What I'm trying to do is FOR EACH person

that ALEC knows, ALEC here,
to each of the different variables here,
and then looking at different things from
each variable as well.
Now I'm going further up the screen
to remind myself what I've already done.
I also want to go back in the book
to find out where I can't...
whether I can have a check within a check
procedure
because I don't think I can.
flicks through book
I'm not finding the answer.
types
I'm still just sort of thinking here
and trying all sorts of ideas
that go through my head.
That's not going to work.
I probably shouldn't be starting this way.
I should try to think of a different way
to start
but it doesn't seem to make any difference.
S1 has typed:

```
SOLO: TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: CHECK * --- SELLS ---> VIDEOS
.....DONE
.....1AB IF ABSENT: EXIT
...: DONE
```

reads screen
types
OK. Back in the book.
types
I've made a mistake here. CHECK...
I'll just write DONE (?) the thing again
S1 has written:

```
SOLO: TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: FOR EACH CASE OF * --- SELLS ---> ?B
```

.....1AA: *B --- SELLS ---> VIDEOS

...: DONE

S1 types:

SOLO: TO INVESTIGATE /X/

...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?

.....1A: FOR EACH CASE OF * --- SELLS ---> ?A

.....1AA: CHECK * --- SELLS ---> VIDEOS

.....1AAA IF PRESENT: FOR EACH CASE OF * --- SELLS ---> VIDEOS

.....1AAAA: CHECK * --- HAS ---> LOTSOFMONEY

.....1AAAAA IF PRESENT: NOTE * --- ISA ---> SUSPECT

.....1AAAAB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT

.....1AAB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT]

I'm just reading what I've got on the screen
to see if it makes any sort of sense at all.
(*reads screen*)
Now I'm looking at what I've got here
and I think I've got confused with my asterisks
and my asterisks and suffixes.
OK... And I have done in fact...
My substep 1AA is incorrect
because it's got an A in it too many.
So I just want to have DONE
and then and then I'm going to edit it.
OK. EDIT INVESTIGATE
and then I'm going to say 1AA
OK, let me just check . Asterisk.
OK. So we just CHECK *.
types
OK. I don't know why it's done this
(*editing fails S1 retypes INVESTIGATE*)
SOLO: EDIT INVESTIGATE
...: 1AA: CHECK * --- SELLS ---> VIDEOS
;;; ERROR: is not an external symbol of package 1AA found by reader.
I'm just going to copy it, in that case.
(*S1 tries copying and pasting but makes a number of errors)*
Right at the moment I'm just having problems
making the thing work again.

[*S1 has typed*:

```
SOLO: EDIT INVESTIGATE
...: 1AA: CHECK * --- SELLS ---> VIDEOS
;;; ERROR:  Is not an external symbol of package 1AA found by reader.
>  (solo-top-level)
SOLO: TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A:  FOR EACH CASE OF * --- SELLS ---> ?A
.......1AA: CHECK * --- SELLS ---> VIDEOS
...:
;;; ERROR: OOPS... FOUND NIL INSTEAD OF A LINE NUMBER OR DONE
>  (solo-top-level)
SOLO: to investigate /x/
...: 1 FOR EA
.....1A: EXIT
...: DONE
OK... I NOW KNOW HOW TO 'INVESTIGATE' /X/
SOLO: TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: FOR EACH CASE OF * --- SELLS ---> ?A
.......1AA: CHECK * --- SELLS ---> VIDEOS
.........1AAA IF PRESENT: FOR EACH CASE OF * --- SELLS ---> VIDEOS
...........1AAAA: CHECK * --- HAS ---> LOTSOFMONEY
.............1AAAAA IF PRESENT: NOTE * --- ISA ---> SUSPECT
.............1AAAAB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
.........1AAB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
...: DONE
;;; ERROR: Attempt to read past end of stream.
>  (solo-top-level)
SOLO: to investigate /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: FOR EACH CASE OF * --- SELLS ---> ?A
.......1AA: CHECK * --- SELLS ---> VIDEOS
.........1AAA IF PRESENT: FOR EACH CASE OF * --- SELLS ---> VIDEOS
...........1AAAA: CHECK * --- HAS ---> LOTSOFMONEY
.............1AAAAA IF PRESENT: NOTE * --- ISA ---> SUSPECT; EXIT
.............1AAAAB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
.........1AAB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
...: DONE
```

```
OK... I NOW KNOW HOW TO 'INVESTIGATE' /X/
SOLO: INVESTIGATE ALEC
OK... (I ALREADY KNEW PETE --- ISA ---> SUSPECT)
SOLO: TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: CHECK * --- SELLS ---> VIDEOS
.......1AA IF PRESENT: FOR EACH CASE OF * --- SELLS ---> VIDEOS
.........1AAA: CHECK * --- HAS ---> LOTSOFMONEY
...........1AAAA IF PRESENT: NOTE * --- ISA ---> SUSPECT; NEXT CASE
...........1AAAB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
.......1AB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
...: DONE
OK... I NOW KNOW HOW TO 'INVESTIGATE' /X/
SOLO: EDIT INVESTIGATE
...: 1AAA: CHECK * --- HAS ---> LOTSOFMONEY
;;; ERROR: Package 1AAA does not exist.
>
 TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: CHECK * --- SELLS ---> VIDEOS
.......1AA IF PRESENT: FOR EACH CASE OF * --- SELLS ---> VIDEOS
.........1AAA: CHECK * --- HAS ---> LOTSOFMONEY
...........1AAAA IF PRESENT: NOTE * --- ISA ---> SUSPECT; NEXT CASE
...........1AAAB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
.......1AB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
...: DONE
;;; ERROR: Unbound variable to in eval
>  (solo-top-level)
SOLO: TO INVESTIGATE /X/
...: ...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: CHECK * --- SELLS ---> VIDEOS
.......1AA IF PRESENT: FOR EACH CASE OF * --- SELLS ---> VIDEOS
.........1AAA: CHECK * --- HAS ---> LOTSOFMONEY
...........1AAAA IF PRESENT: NOTE * --- ISA ---> SUSPECT; NEXT CASE
...........1AAAB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
.......1AB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
...: DONE
;;; ERROR: Token composed solely of dots found by reader: ...
>  (solo-top-level)
```

```
SOLO: TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: CHECK * --- SELLS ---> VIDEOS
;;; ERROR: Attempt to read past end of stream.
>  (solo-top-level)
SOLO: TO INVESTIGATE /X/
...:1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
;;; ERROR: Attempt to read past end of stream.
>  (solo-top-level)
SOLO: TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A:CHECK * --- SELLS ---> VIDEOS
;;; ERROR: Attempt to read past end of stream.
>  (solo-top-level)
SOLO: TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: CHECK * --- SELLS ---> VIDEOS
.......1AA IF PRESENT: FOR EACH CASE OF * --- SELLS ---> VIDEOS
.........1AAA: CHECK * --- HAS ---> LOTSOFMONEY
...........1AAAA IF PRESENT: NOTE * --- ISA ---> SUSPECT; NEXT CASE
...........1AAAB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
.......1AB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
...: done
OK... I NOW KNOW HOW TO 'INVESTIGATE' /X/
SOLO: INVESTIGATE ALEC
OK... (I ALREADY KNEW PETE --- ISA ---> SUSPECT)
OOPS... Line 1A within CHECK must end with CONTINUE or EXIT
SOLO: TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: FOR EACH CASE OF * --- SELLS ---> VIDEOS
.......1AA: CHECK * --- HAS --->LOTSOFMONEY
.........1AAA IF PRESENT: NOTE * --- ISA --->SUSPECT; NEXT CASE
.........1AAB IF ABSENT: PRINT * "IS ELIMINATED"; NEXT CASE
...: DONE
OK... I NOW KNOW HOW TO 'INVESTIGATE' /X/]
```

Now I'm just reading what I've got on the

screen

to see if it will work so far.

reads solution

And I'm looking now... back at the book
for the NEXT CASE thing still.
I'm looking at page 88 at the example,
the BOOZETEST example.
reads example
No this has just got me right back to where
I started from at the beginning...
recursing.
reads
I assumed in the last example
that I couldn't have a substep which said
IF PRESENT FOR EACH CASE OF whatever,
blah, blah, blah, blah, semi-colon,
and either a control statement.
I'm assuming I can't have a control statement
after a FOR EACH construct within a sub-step.
Although I don't actually know if that's true
because it... it doesn't say so anywhere.
reads solution
Is it going to make any difference if I have a
CONTINUE?
(?) for the moment just to get out of it.
I'm just keeping on going in circles.
My last thing that I did worked to the same
extent
as the one before that did,
in other words it noted PETE is a SUSPECT
but it didn't do any of the printing of the rest
of the things business.
Because, I'll show you why, will I?
See, here's what it was:
FOR EACH CASE OF that
CHECK * SELLS VIDEOS IF PRESENT;
and then I had a FOR EACH THING
after an IF PRESENT, right?
And it didn't accept it,
because, of course, I have to have a control
statement after it
if there's an IF PRESENT.

And I'm assuming that I can't have a control
statement after a FOR EACH construct.
Is that a correct assumption to be making.
E: (?) Well, you ought to have it down
about here. Because that's the end of the
IF PRESENT (*indicates end of 1AAAB*)
S1: Where would I have to have it?
Here?
E: You've got IF PRESENT blah, blah, blah
S1: It would have to be here. EXIT; EXIT.
E: I don't think that would work at all.
It wouldn't make sense.
S1: I thought that something like this
was going to be the way into solving it
and that's really what I've been working on
for the past hour.
Forgetting...
well not... not at all taking account of the fact
that I was going to have a control statement
at the end of this IF PRESENT thing.
So none of that's going to work.
It accepts an awful lot of things
and then it doesn't actually tell you
until you try to run the procedure
that it hasn't accepted them.
I found this quite often, I mean,
in... in the whole of my way through the
SOLO thing
that it accepts things that are entirely wrong
and doesn't tell you that they're wrong
until you try to run the procedure.
I'm really annoyed that I can't do this.
E: Do you want to give up for the moment?
S1: Uhuh.

In informal discussion after this episode the experimenter asked why the subject gave up on the idea of using a procedure within a procedure. The subject again claimed that such a method did not seem 'elegant' and was determined to find a way to solve the problem within one procedure. The subject then proceeded to solve the problem with paper and

pencil, writing down a solution immediately without reference to any prior solution or the textbook. The solution produced was:

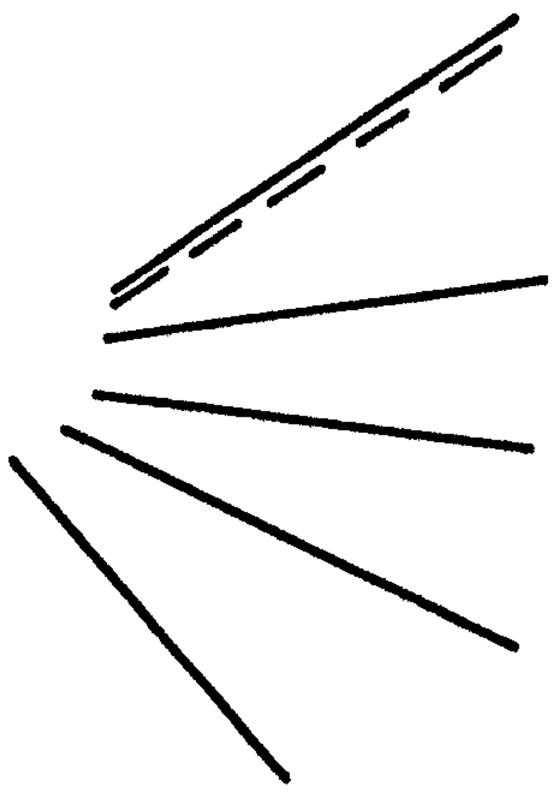
SOLO: TO INVESTIGATE /X/
...: 1 FOR EACH CASE OF /X/ --- KNOWS ---> ?
.....1A: CHECK * --- SELLS ---> VIDEOS
.....1AA IF PRESENT: SUSS *; NEXT CASE
.....1AB IF ABSENT: PRINT * "IS ELIMINATED"; EXIT
...: DONE

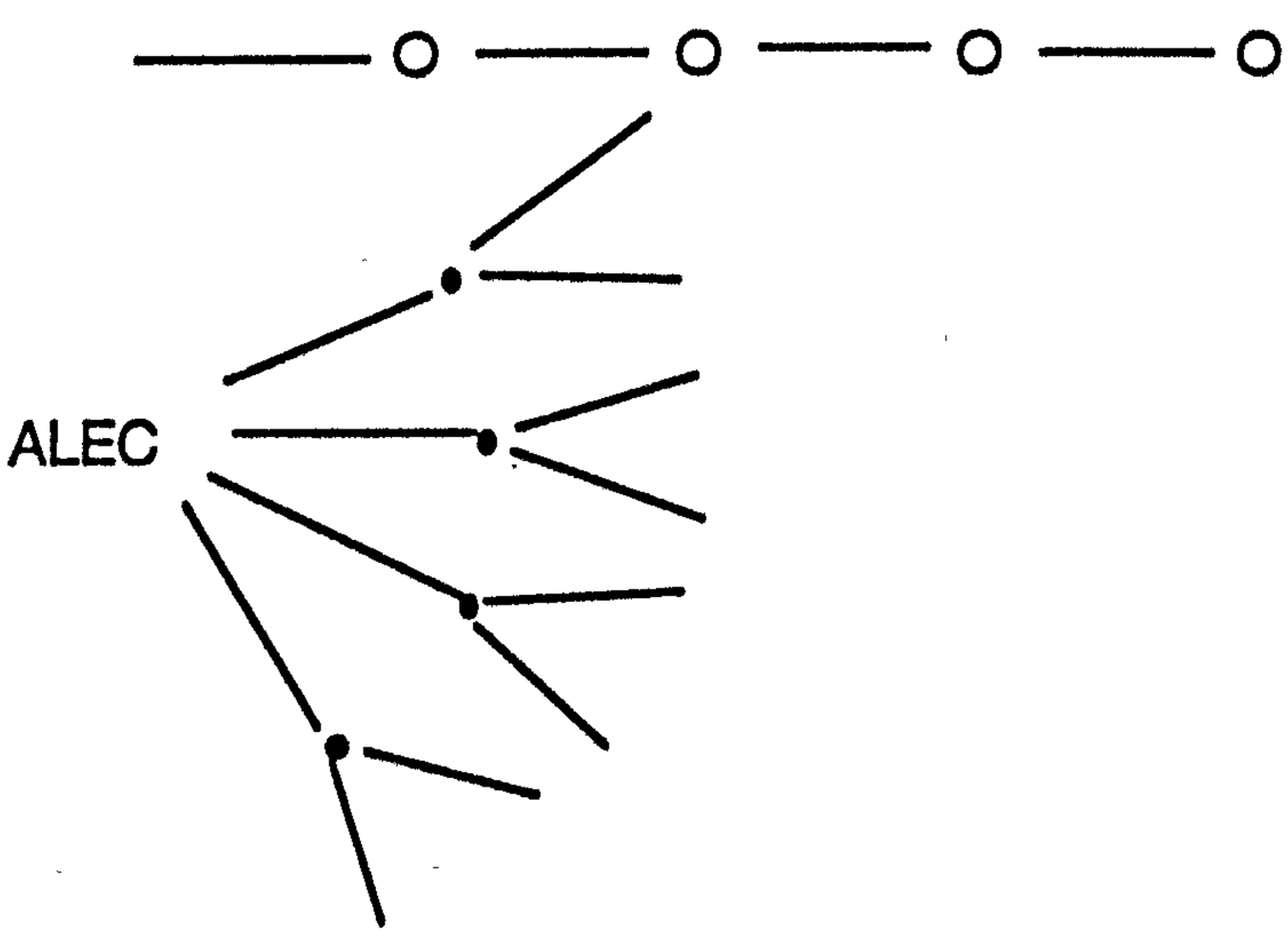
OK... I NOW KNOW HOW TO 'INVESTIGATE' /X/

SOLO: TO SUSS /X/
...: 1 CHECK /X/ --- HAS ---> LOTSOFMONEY
.....1A IF PRESENT: NOTE /X/ --- ISA ---> SUSPECT; EXIT
.....1B IF ABSENT: PRINT /X/ "IS ELIMINATED"; EXIT
...: DONE

This solution would not go through the whole database since the subject has written 'EXIT' at the end of line 1AB in the INVESTIGATE procedure. The subject was aware of this, however, pointed out to the experimenter that she was unsure whether it should be EXIT or NEXT CASE then changed it to NEXT CASE .

S1'S DRAWINGS:




ALEC

A.3.5 Think aloud protocol from S1: Solving the third SOLO problem

E: 14th February. Problem 3.

S1: How to spend your Valentine's Day.

(reads first line of problem)

'Create a database containing the following

information.'

Right, well, I'll just go ahead and do that.

types in database

OK. The problem, 'Now write a problem

called INVITE which NOTEs that....'

This is nice. What a nice sociable sort of

problem. (?)

Right.

Re-reads problem

So BILL LIKES ISOBEL but ISOBEL doesn't

like BILL? Is that right?

E: That's the only database you've got.

S1: Yes, OK. Yes, OK,

but I mean that... those...

I mean, for instance, AVRIL LIKES BILL

but BILL doesn't like AVRIL

but these things aren't necessarily important

for the thing.

I mean, they're not some sort of deliberate

things?

OK. Em, Right.

Well I'm just checking back to the SOLO

book here.

Em, just for fun.

Page 91. Page 92.

And I think this is going to be pretty similar

to the INFECT thing on page 92, really,

isn't it?

So the way I'm going to deal with this....

I'm looking back to page 91.

I'm looking back at the bit called A.7

'Iteration versus Recursion' and again this

little fan-shaped diagram indicating
iteration
and the line-shaped thing for recursion
just... because I like them.
OK, so I think the easiest way to do this is
just to start typing things
and figure out where they're not going to
find... and...
em, what did I just say there?
I'm going to start typing and find out
once I'm sort of half way through it
why it's not going to work. OK?
So we start off..
(to E) Well I've got to say something.
E: That's fine.
S1: *types*
SOLO: TO INVITE /X/
...: 1 NOTE /X/ --- HAS ---> INVITATION
...: 2 FOR EACH CASE OF /X/ --- LIKES ---> ?
.....2A: INVITE *
Right, I'm just thinking now.
Reads out what she has just typed
Now, is that going to work?
Why don't I type it in and find out?
So, that it's got iteration and recursion in it.
E: Are you reading through it again to see
if it's going to work?
S1: No, I'm just reading through it again.
I can't be bothered going through...
What I should really do if I had any sense
em, is actually pick one of these names
and stick it in as an example.
But I can't be bothered
so I'm just going to type it in
and see what it comes up with.
types DONE
[program now is:
SOLO: TO INVITE /X/
...: 1 NOTE /X/ --- HAS ---> INVITATION

...: 2 FOR EACH CASE OF /X/ --- LIKES ---> ?

.....2A: INVITE *

...: DONE

OK... I NOW KNOW HOW TO 'INVITE' /X/

S1 now types:

SOLO: INVITE AVRIL

SOLO responds:

OK...

AVRIL

|

| --- LIKES ---> BILL

|

| --- LIKES ---> ILONA

|

| --- LIKES ---> JOHN

|

| --- LIKES ---> KAREN

|

| --- HAS ---> INVITATION

OOPS... THE FORMAT FOR 'FOR' IS:

FOR EACH CASE OF FROM ---

RELATION ---> VALUE]

S1: Right, I've obviously just...

I've obviously, em, missed...

E: Go back up to your...

S1: I've missed a space between the end of

the arrowhead and the question mark.

That's what it is, isn't it?

E: Yes, that's it. You can...

That's easy to edit at least.

[S1 types:

SOLO: EDIT INVITE

...: 2 FOR EACH CASE OF /X/ --- LIKES ---> ?

.....2A: INVITE *

...: DONE

SOLO responds:

OK... I HAVE RE-DEFINED HOW TO 'INVITE' /X/

S1 types:

SOLO: INVITE AVRIL

SOLO responds:

```
OK... (I ALREADY KNEW AVRIL --- HAS ---> INVITATION)
OK...
   BILL
     |
     | --- LIKES ---> CAROL
     |
     | --- LIKES ---> ISOBEL
     |
     | --- HAS ---> INVITATION
OK...
   CAROL
     |
     | --- LIKES ---> EMMA
     |
     | --- LIKES ---> FRANK
     |
     | --- LIKES ---> GRAHAM
     |
     | --- HAS ---> INVITATION
OK...
   EMMA
     |
     | --- HAS ---> INVITATION
OK...
   FRANK
     |
     | --- LIKES ---> GRAHAM
     |
     | --- HAS ---> INVITATION
OK...
   GRAHAM
     |
     | --- HAS ---> INVITATION
OK... (I ALREADY KNEW GRAHAM --- HAS ---> INVITATION)
```

```
OK...
   ISOBEL
     |
     | --- LIKES ---> HANNAH
     |
     | --- HAS ---> INVITATION
OK...
   HANNAH
     |
     | --- HAS ---> INVITATION
OK...
   ILONA
     |
     | --- HAS ---> INVITATION
OK...
   JOHN
     |
     | --- LIKES ---> LOTTE
     |
     | --- HAS ---> INVITATION
OK...
   LOTTE
     |
     | --- HAS ---> INVITATION
OK...
   KAREN
     |
     | --- HAS ---> INVITATION]
```

S1: Is that it ?

E: That's it.

S1: I'm finished?

E: Yes.

S1: Gosh.

S1: Right we're discussing why this one wasn't harder than the previous one, yeah?

E: Yes, this one involves both recursion and iteration.

S1: Yeah, it's exactly the same as the example on page 92.

E: Yes, but you had to find that out.

S1: Yes, I know, but I had done the example on page 92 and I remembered it, and, I mean, I read through the thing before I did the exercise, although I remembered it anyway. I could have done that without looking at page 92. The difficulty with the second problem was that I was having to do things that hadn't ever come out in the book. Em, well I mean, I wasn't really, because, I mean, the thing was solved quite simply by having two constructs, no, two procedures. Yes, that's it. By having two procedures running simultaneously. Well, actually not running simultaneously but one running inside the other.

E: Yeah, one calling the other.

S1: Yes, but that wasn't something that was ever particularly used in the book as an example. Well, I mean that's a silly sort of thing to say, because it was very straightforward. The reason it took me so long was because we discussed this as well after it. Do you want me to say all this now?

E: Yes, yes.

S1: Well, as we discussed after I had actually solved the second one, I said quite early on in my solving of it, or my attempts to solve it, that it would work perfectly well very easily with two procedures going. But the reason I didn't go ahead and solve it then, was because I wondered ... I though it might be possible to solve it by just having one procedure which would, of course, be more complicated than having two ones. It would be a more complicated procedure than either of the two that were required to solve it. But I thought it would be quite fun... or something ...

E: You said you thought it would be more elegant do it in one.

S1: Yes, something like that. I thought it would be neater if it were possible to have simply one procedure. And I wanted to see exactly how all the deeply embedded things worked, and, of course, I found eventually that I couldn't do it, and once I decided to go back and do it using two procedures, as you recall, I basically sat down and wrote them down immediately without even thinking about it, and it was very straightforward.

E: Right this is the only... this is the section of the book that has the... that calls another one. (indicates page 56)

S1: Yes, I know. It does.

E: It's just that it's quite a bit back, this is page 56.

S1: Yes, I mean, I do remember that. It was just... yes I mean... I mean, as I think we... we established as I was doing the thing, I did work out that it would be possible to do it with two procedures. It's just that I wanted to see if I could do it with one.

E: Stubbornness.

S1: Something like that.

E: OK. Thanks very much.

A.3.6 Think aloud protocol from S2: Solving the first SOLO problem

E: Session 1, problem 1.

This is the problem. [*hands sheets to S2*]

S2: All right.

E; If you could just read it through then...

solve it.

S2: *reads problem.*

E: Also, if you re-read any of it, could

you say which bits you're re-reading.

S2: Rereading the first page.

reads first page

Reading second page.

reads second page

E: Can I ask you what are you're

immediate thoughts?

S2: Em... First off the problem looks

relatively...

when I say relatively... it looks fairly easy.

Em... but on reflection to that,

you obviously...

you've got to go through and work out

exactly what the problem's asking you...

em... in respects of solving it.

Em... So I believe...

to go through it and

check all the actual, em... database entries

for the, em... em... the node of OVER...

if... if it finds OVER

then to include IS FLOODED on it.

E: Right.

rereads problem statement

S2: Where's the book?

flicks through book

E: Which... have you found a page yet?

S2: Not yet.

reads problem statement

E: Are you able to say what it is that

you're thinking about?
S2: Em... I'm just trying to work out
how the actual program's gonna start
working through
and how to branch it out.
Em.. So I think the first thing I want to do
is probably to, em...
write FLOOD
and first off to, em... FLOOD ATTIC, em...
Then do a CHECK to
check that is OVER, em... a particular room,
and work that through
and FLOOD all those various rooms.
flicks through book
I trying to work out the best... thing to do.
reads problem statement
flicks through book
E: If you actually read anything from the
book, just say which one you're reading.
The name of the example will do.
S2: Right. Okay. Em...
There are going to be a few ways
of doing this.
Do you want the quickest, longest, or any
other, or just to solve the problem?...
E: Any way that'll give you that in the
database.
[*referring to final database after program has run*]
S2: Right. Okay.
Reading page 67, top right hand example.
reads example and refers to problem
statement several times
E: Can I ask you to tell me how you think
the example is going to help you? Or if it is?
S2: Em... Just trying to see if I can, em...
this is just using CHECK statements.
Em...
E: So at the moment you're just looking at
the CHECK statements.

S2: Yeah, yeah, em...
seeing if it's going to help me at all.
This probably will work,
it's going to be incredibly long.
Em... It might be better to do FOR EACH
CASE statements...
(?)
check to see if I can fit the problem to it.
Probably a quicker solution.
Reading page 88 top left hand example.
reads example
reads problem statement
It's probably not going to work.
So go back to the CHECK statements.
On 67 again.
Okay, I can probably get this to work.
Look at the, em... results.
reads page 2 of problem statement
He says hoping.
Em... Right...
types
spell FLOOD.
Subject has written:
TO FLOOD /X/
1 CHECK /X/ OVER ?
rereads problem statement
E: What are you...?
S2: I'm just...
the first one.
Gonna put in CHECK statements
to check if the em...
ATTIC is OVER statement.
Em... obviously, if I do that,
it's not going to come up with the, em...
database of ATTIC IS FLOODED.
reads partial solution
Em... or will it?
E: Can I ask you if you're thinking of a way
of getting round it, or whether you're thinking of...

S2: I'm thinking of scrapping this
and doing it again.
types return.
[SOLO crashes and asks whether to abort]
E: It's probably best to... abort in this case.
It's not a very friendly program.
Solo-top-level.
S2: *restarts SOLO*
rereads problem statement and example
several times
E: Are you reading the problem statement?
S2: Yeah.
E: I'm going to make comments like that
every now and again.
S2: Right, right.
Seeing if I can think this over.
If my inspiration comes to life.
rereads problem statement
E: What is it you're thinking about?
How to get...?
S2: Getting past...
Trying to work out the structure of FLOOD.
E: Right.
S2: Em...
rereads problem statement
E: Can you say what it is you think FLOOD
needs?
S2: Em... Probably CHECK statements
to check, em...
the first one is FLOOD the ATTIC,
so I'd CHECK the ATTIC.
First I want to FLOOD the ATTIC
so I can put an inclusion
on the ATTIC IS FLOODED.
Then CHECK the ATTIC is OVER a room,
then FLOOD those various rooms.
E: Right.
S2: *types*
E: So you've written NOTE /X/ --- IS --->

FLOODED; 2 CHECK /X/ OVER ?.

S2: Uhuh.

rereads problem statement and solution

E: Can I ask you what you're thinking now?

S2: I'm thinking now...

It's obviously going to come up with

I've FLOODED the ATTIC,

now I'm checking whether the ATTIC is OVER.

Trying to then work out, em...

to then FLOOD that.

E: So in other words you're going through

what you've written with ATTIC in the place

of /X/ at the moment, just to see how it works.

Right?

S2: Yeah.

reads solution

types

E: So you've got '2a IF PRESENT: NOTE *

IS FLOODED'

S2: Yeah.

E: Can I ask you what you're thinking?

S2: Just trying to work out how (?)...

I'm trying now to do a CHECK on (?) one,

which I can't do.

rereads problem statement

E: Can I ask you if you can recognize what

kind of problem this is?

S2: Em... A hard one.

Em... I don't know.

I'm not doing very well on the first one.

Reading page 70 at the moment

E: Is that the new CRAVETEST?

S2: Yeah.

E: Could you say which bits, em.. you're

reading? Is it the text or the example?

S2: The example.

E: Are you looking for anything specifically?

S2: I'm just trying to remember an example

I've already done

which I can then... tie this into.

Em... If I can find it.

E: Can you remember the name of the example?

S2: Em...

E: Or something about it?

S2: *I believe it's the one where, em...*

I think it's the INSULT one.

You define two programs, em...

E: I think INSULT is (?). (*looks up book*) 39?

S2: I think, if that's the case,

I think it's the INSULT and PRAISE.

They're em...

It's em...

E: Page 57 is it?

S2: Yeah.

E: To JUDGE?

S2: Yeah.

reads solution

reads example

E: Are you trying to see how that one would help?

S2: Yeah, I'm just trying to see

if this one would help.

Em...

I was trying to...

make it very simpler,

would be simply to say NOTE em...

it's *FLOODED.*

Would make it something like a subprogram.

Something like FLOOD1...

em... X or something,

and then FLOOD the various rooms (?) em...

lots of fun doing that.

reads manual

E: Page 80.

S2: Yeah. INFECT.

reads example

skims through book

reads new version of INFECT (page 92)

E: And this is page 92. And what do you think about that one?

S2: Em... Looks like it may... well... do it. Em...

E: Can you say why you think it might do?

S2: Em... Not at the moment. Em...

E: Can I ask you also, did you read the text round about it? Or just...

S2: Yeah. I'm reading the text now. I'm referring back to the example

reads example

'the INFECT procedure is defined in section A.1 can now be defined as follows'

E: Page 80

S2: Right. I'm going back to that.

E; And you're reading the text here as well, I see.

S2: Yeah.

reads text

Just going to, em...

types

Oops. Abort.

types

E: Probably be simpler typing it again, to be honest.

S2. *reads problem*

types

E: Em... Arrowhead.

S2: Oops.

E: Is there a space there.

S2: Can I use 'DONE' now to get out of that?

E: No. Not there. EXIT

S2: *types*

E: So you've got: TO FLOOD /X/ NOTE /X/ IS FLOODED; 2 CHECK /X/ OVER ?; 2A IF PRESENT: FLOOD *;EXIT;

2B; EXIT; DONE.

S2: Let's try.

types:

FLOOD ATTIC

[*SOLO responds:*

OK...

ATTIC

|

| --- OVER ---> BEDROOM

|

| --- IS ---> FLOODED

OK...

BEDROOM

|

| --- ISA ---> ROOM

|

| --- OVER ---> LANDING

|

| --- IS ---> FLOODED

OK...

LANDING

|

| --- ISA ---> PLATFORM

|

| --- OVER ---> CUPBOARD

|

| --- IS ---> FLOODED

OK...

CUPBOARD

|

| --- OVER ---> LOUNGE

|

| --- IS ---> FLOODED

OK...

LOUNGE

|

| --- ISA ---> ROOM

|

| --- OVER ---> KITCHEN

```
  |
   | --- IS ---> FLOODED
OK...
 KITCHEN
  |
   | --- ISA ---> ROOM
  |
   | --- OVER ---> BASEMENT
  |
   | --- IS ---> FLOODED
OK...
 BASEMENT
  |
   | --- HAS ---> CONCRETEFLOOR
  |
   | --- IS ---> FLOODED]
```

E: Right.

S2: What a nightmare.